Collection

*Ressources Informatiques*

# PHP et MySQL

## Maîtrisez le développement d'un site Web dynamique et interactif

Olivier HEURTEL

INFORMATIQUE TECHNIQUE

eni éditions

# PHP et MySQL

**Maîtrisez le développement d'un site Web dynamique et interactif**

Olivier HEURTEL




## Résumé

Ce livre sur **PHP et MySQL** s'adresse aux concepteurs et développeurs qui souhaitent utiliser PHP et MySQL pour développer un site Web dynamique et interactif.
Dans la **première partie** du livre, l'auteur présente la **mise en oeuvre d'une base de données MySQL** : langage SQL (Structured Query Language), utilisation des fonctions MySQL, construction d'une base de données (tables, index, vues), sans oublier les techniques avancées comme la recherche en texte intégral ou le développement de programmes stockés.
Dans la **deuxième partie** du livre, après une présentation des **fonctionnalités de base du langage PHP**, l'auteur se focalise sur les **besoins spécifiques du développement de sites dynamiques et interactifs** en s'attachant à apporter des réponses précises et complètes aux problématiques habituelles : gestion des formulaires, gestion des sessions, envoi de courriers électroniques et bien sûr accès à une base de données MySQL.
Abondamment illustré d'exemples commentés, ce livre (écrit sur les versions 5 de PHP et de MySQL) est à la fois complet et synthétique et vous permet d'aller droit au but.
Les exemples cités dans le livre sont en téléchargement sur cette page.

## L'auteur

Après plus de huit ans passés en société de service, où il a successivement occupé les postes de développeur, chef de projet puis directeur de projet, **Olivier Heurtel** a démarré une activité de consultant/formateur indépendant spécialisé sur les bases de données (Oracle), le développement Web (PHP) et les systèmes décisionnels. Il est certifié Oracle Certified Professional.

# Objectifs de l’ouvrage

L’objectif de cet ouvrage est d’apprendre à développer un site Web dynamique et interactif à l’aide de PHP et MySQL.

Pour répondre à cet objectif, ce livre étudie en détail les fonctionnalités nécessaires au développement d’un site Web dynamique et interactif :

- utilisation du langage SQL (*Structered Query Language* - langage standard d’accès aux bases de données relationnelles) pour interroger et modifier les données d’une base de données MySQL ;
- création d’une base de données MySQL ;
- utilisation des fonctionnalités de base du langage PHP ;
- accès à une base de données MySQL à partir de PHP ;
- gestion des formulaires ;
- gestion des sessions (authentification, gestion d’un contexte, utilisation des cookies) ;
- envoi d’un courrier électronique (dont les courriers au format HTML et ceux avec pièce jointe) ;
- gestion des fichiers, dont le transfert de fichiers du poste de l’utilisateur vers le serveur ("file upload").

Cet ouvrage s’adresse à des chefs de projet, concepteurs ou développeurs ayant une connaissance de base de la programmation Web en HTML.

Ce livre aborde les versions 5 de PHP et de MySQL.

# Bref historique de PHP et MySQL

## 1. PHP

Le langage PHP (historiquement *Personal Home Page*, officiellement acronyme récursif de *PHP : Hypertext Preprocessor*) a été conçu en 1994 par Rasmus Lerdorf pour ces besoins personnels, avant d'être rendu public au début de l'année 1995.

Courant 1995, une nouvelle version, complètement réécrite, est publiée sous le nom PHP/FI version 2. Cette version, capable de gérer les formulaires et d'accéder à la base mSQL, permet au langage de se développer rapidement.

En 1997, le développement du langage est pris en charge par une équipe formée autour de Rasmus Lerdorf et aboutit à la sortie de la version 3.

En 2000, l'analyseur PHP est migré sur le moteur d'analyse Zend afin d'offrir de meilleures performances et de supporter un plus grand nombre d'extension : c'est la version 4 de PHP.

En 2004, la version 5 voit le jour. Cette nouvelle version, basée sur la version 2 du moteur Zend, apporte plusieurs nouveautés, la plupart concernant le développement orienté objet.

À ce jour, les analystes estiment que PHP est utilisé par plus de 20 millions de sites Web dans le monde (en nombre de domaines).

## 2. MySQL

MySQL est le *Système de Gestion de Base de Données Relationnelle* (SGBDR) Open Source le plus répandu dans le monde. Il est développé par MySQL AB, une entreprise suédoise.

La première version de MySQL est apparue en 1995. Cette première version est créée pour un usage personnel à partir de mSQL.

En 2000, la version 3.23 est passée en licence GPL (*General Public License*).

En 2003, la version 4, apparue en 2001, est déclarée stable. Cette version apporte de nombreuses nouvelles fonctionnalités et améliorations : opérateur `UNION`, `DELETE` multi-tables, nouvelles options pour la gestion des droits, amélioration des performances, sous-requêtes (4.1), etc.

En 2005, la version 5, apparue en 2003, est déclarée stable. Cette version majeure introduit de nombreuses fonctionnalités manquantes dans MySQL : programmes stockés, triggers, vues.

Fin 2007, la version 5.1 est distribuée en Release Candidate et devrait sortir en version finale début 2008.

MySQL est disponible selon deux licences différentes :

- La licence GPL ;
- Une licence commerciale.

Si vous utilisez MySQL dans un produit libre, vous pouvez utiliser MySQL librement (version MySQL Community Server). Si vous utilisez MySQL dans un produit commercial, ou si vous souhaitez avoir un support pour le logiciel, vous devez acquérir une licence commerciale (version MySQL Enterprise).

# Où se procurer PHP et MySQL

De nombreux sites Web sont consacrés au langage PHP et à MySQL. Ils permettent de télécharger les produits, de consulter des exemples de scripts ou de dialoguer sur des forums :

| Adresse | Contenu |
|---|---|
| www.php.net | Site officiel de PHP qui propose le téléchargement de PHP et un manuel de référence en ligne très pratique. Vous pouvez notamment saisir `www.php.net/nom_fonction` pour accéder directement à l'aide en ligne d'une fonction PHP. |
| www-fr.mysql.com | Site officiel de MySQL qui propose le téléchargement de MySQL, une aide en ligne, des articles, un forum, etc. |
| www.phpindex.com | Site en français consacré à PHP qui propose des news, des exemples ainsi qu'un forum de discussion. Bref, un site très complet à mettre dans ses favoris. |
| www.phpfrance.com | Autre site francophone consacré à PHP proposant des rubriques similaires. |
| www.zend.com/fr | Site officiel du moteur de script Zend qui propose lui aussi les rubriques classiques de téléchargement, d'exemples, de forum, … |
| www.easyphp.org | Site francophone qui propose gratuitement un produit installable (EasyPHP) sur plate-forme Windows. Ce produit comprend : un serveur Apache, PHP et MySQL. Vous téléchargez le produit et double cliquez sur l'exécutable qui installe les différents éléments. Cinq minutes après, votre environnement PHP-MySQL est opérationnel. Ce site est indispensable pour ceux qui souhaitent monter rapidement une configuration opérationnelle complète sur Windows. Les versions utilisées par EasyPHP présentent toujours un léger retard par rapport aux dernières versions officielles. |
| www.apachefriends.org/fr/xampp.html | Autre site qui propose un produit installable (XAMPP) sur différentes plates-formes (Linux, Windows, Solaris, Mac OS X). Ce produit comprend lui aussi, entre autres, un serveur Apache, PHP et MySQL. Là encore l'installation est très simple et très rapide. Les versions utilisées par XAMPP sont toujours très récentes par rapport aux dernières versions officielles. |
| www.editions-eni.com/exemples/ | Page du site des Éditions ENI sur laquelle les exemples traités dans cet ouvrage peuvent être téléchargés. |

Cette liste est évidemment non exhaustive mais tous les sites présentés proposent de nombreux liens vers d'autres sites. N'hésitez pas à surfer !

Tous les exemples de cet ouvrage ont été testés avec XAMPP Linux 1.6.4 (soit MySQL 5.0.45 et PHP 5.2.4).

# Conventions d'écriture

## 1. PHP

La syntaxe des fonctions PHP est décrite de la manière suivante dans cet ouvrage :

```
type_retour nom_fonction(type_paramètre nom_paramètre)
```

*type_retour*

Type de retour de la fonction.

nom_fonction

Nom de la fonction.

*type_paramètre*

Type du paramètre accepté par la fonction.

nom_paramètre

Nom donné au paramètre.

Les types de données possibles seront présentés dans le chapitre Introduction à PHP. Dans le cas où la fonction accepte un paramètre de n'importe quel type et/ou retourne une valeur de n'importe quel type, le terme *mixte* est utilisé.

Si la fonction ne retourne pas de valeur, l'information *type_retour* est omise.

*Exemple*

```
nom_fonction(type_paramètre nom_paramètre)
```

Si la fonction ne prend aucun paramètre, les informations *type_paramètre* et nom_paramètre sont omises.

*Exemple*

```
type_retour nom_fonction()
```

Les paramètres optionnels sont indiqués entre crochets ([]).

*Exemple*

```
type_retour nom_fonction([type_paramètre nom_paramètre])
```

Si la fonction accepte plusieurs paramètres, ces derniers sont indiqués, séparés par une virgule, selon la même convention.

*Exemple*

```
type_retour nom_fonction(type_paramètre_1 nom_paramètre_1, type_paramètre_2
nom_paramètre_2)
```

Si un paramètre peut être répété un nombre quelconque de fois, il est simplement suivi de la séquence [,...].

*Exemple*

```
type_retour nom_fonction(type_paramètre nom_paramètre[,...])
```

## 2. MySQL

La syntaxe des ordres SQL est décrite de la manière suivante dans cet ouvrage :

MOT EN MAJUSCULES

Mots clés de la commande (`CREATE TABLE` par exemple).

Dans la pratique, ils peuvent être saisis indifféremment en majuscules ou en minuscules.

`mot en minucules`

Valeurs à saisir relatives à la base de données ou à l'application (nom de table, nom de colonne, etc).

Selon le cas, ces valeurs sont sensibles à la casse ou pas (cf. Introduction à MySQL - phpMyAdmin).

`[]`

Clause optionnelle.

`[,…]`

La clause précédente peut être répétée plusieurs fois.

`|`

Indique un choix entre plusieurs options.

`{}`

Délimite une liste d'options.

<u>`mot souligné`</u>

Valeur par défaut.

*`mot en italique`*

Clause de la commande dont la syntaxe est détaillée à part.

# Introduction aux bases de données relationnelles

## 1. Concepts

Une base de données est un ensemble de données structurées correspondant généralement à un domaine fonctionnel (facturation, ressources humaines, etc.). Physiquement, une base de données se matérialise par un ensemble de fichiers stockés sur un périphérique de stockage.

Les données d'une base de données sont gérées par un logiciel appelé *Système de Gestion de Base de Données* (SGBD). Ce logiciel offre plusieurs fonctionnalités : accès aux données, gestion des mises à jour, renforcement de l'intégrité, contrôle de la sécurité d'accès, etc.

Une base de données relationnelle supporte une organisation des données basées sur le modèle relationnel, développé en 1970 par Edgar Frank Codd. C'est la structure la plus répandue aujourd'hui.

Dans une base de données relationnelle, les données sont organisées en tables logiquement liées entre elles. Une table comporte un certain nombre de colonnes (ou champs) qui décrivent une ligne (ou enregistrement). La mise en relation des tables s'effectue par l'intermédiaire d'une colonne.

<u>*Exemple*</u>

```
livre
+----+--------------------------------+---------------+
| id | titre                          | id_collection |
+----+--------------------------------+---------------+
|  1 | PHP 5.2 - Développement Web    |             1 |
|  2 | Oracle 10g - Administration    |             1 |
|  3 | Oracle 10g - Recovery Manager  |             2 |
|  4 | BusinessObjects 6              |             1 |
|  5 | MySQL 5 - Mise en oeuvre       |             1 |
|  6 | PHP et MySQL (versions 4 et 5) |             3 |
|  7 | MySQL 5 et PHP 5               |             4 |
+----+--------------------------------+---------------+

collection
+----+-------------------------+
| id | nom                     |
+----+-------------------------+
|  1 | Ressources Informatiques |
|  2 | TechNote                |
|  3 | Les TP Informatiques    |
|  4 | Coffret Technique       |
+----+-------------------------+
```

Sur cet exemple, les tables `livre` et `collection` sont liées par les colonnes `id_collection` de la table `livre` et `id` de la table `collection`.

L'interaction avec une base de données relationnelle s'effectue grâce au langage SQL (*Structured Query Langage*). Ce langage permet la lecture et la mise à jour des données, mais aussi la définition de l'organisation des données, la gestion de la sécurité, le renforcement de l'intégrité, etc. Le langage SQL est un langage normalisé, mais les différents éditeurs de base de données ne respectent pas l'intégralité du standard.

## 2. Principes de conception d'une base de données

La conception d'une base de données est un sujet complexe dont nous n'abordons ici que les principes de base, dans une approche plus pratique que théorique.

Dans une base de données relationnelle, l'objectif est de stocker dans des tables différentes les informations correspondant à des entités (objets) différents du domaine fonctionnel. Le but est d'éviter les redondances et faire en sorte qu'une information donnée ne soit stockée qu'une fois. Sur notre exemple précédent, les informations sur l'auteur d'un livre ne sont pas stockées dans la table livre ; l'auteur d'un livre est une entité fonctionnelle à part entière qui est stockée dans une table séparée.

Ce processus de séparation des données dans plusieurs tables est appelé « normalisation ».

Une normalisation poussée à l'extrême peut nuire aux performances des requêtes d'interrogation qui doivent lire un grand nombre de tables. Pour améliorer les performances des lectures, il est alors envisageable de

« dénormaliser » le modèle, en regroupant des tables, quitte à avoir des données redondantes dans les différentes lignes. Les bases de données des systèmes décisionnels, qui effectuent principalement des interrogations généralement complexes, sont très souvent dénormalisées. À l'inverse, les bases de données des systèmes transactionnels, qui effectuent principalement des petites interrogations simples et beaucoup de mises à jour, respectent bien le principe de normalisation.

---

Dans une base de données relationnelle, chaque table stocke les informations relatives à un objet métier concret ou abstrait qui doit être identifié.

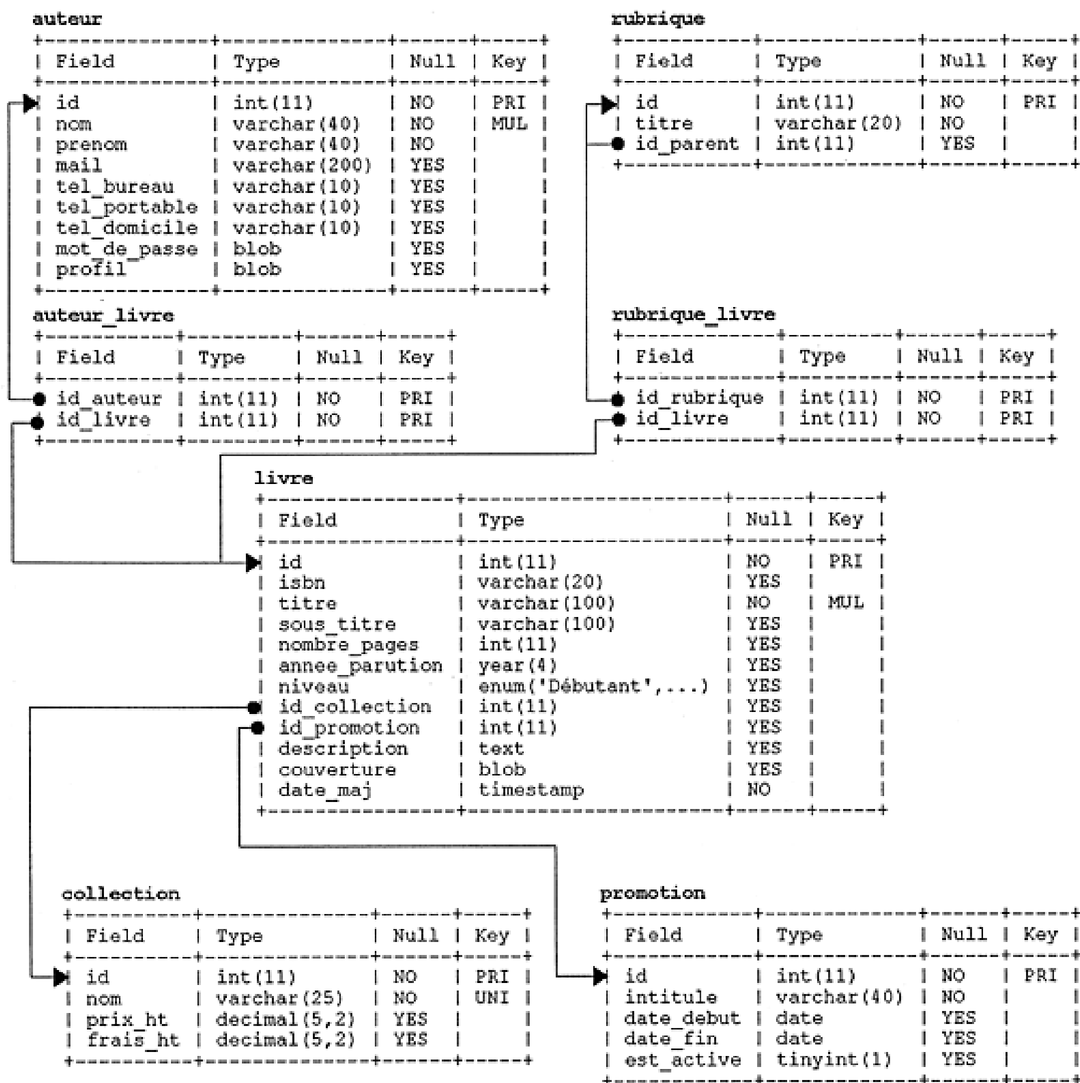
Dans la table, chaque colonne stocke une information unitaire (attribut, propriété) qui caractérise une ligne de la table. Chaque colonne possède un type de données (entier, chaîne de caractères, date, etc) et peut être obligatoire ou non.

Une colonne ou combinaison de colonnes qui identifie de manière unique une ligne d'une table est appelée clé candidate. La valeur d'une clé candidate est différente pour toutes les lignes de la table (pas de doublon autorisé). Une clé candidate peut être constituée par une colonne arbitraire utilisée spécifiquement pour cela.

La clé primaire d'une table est une des clés candidates de la table, choisie plus ou moins arbitrairement, si ce n'est que les colonnes de la clé primaire doivent aussi être obligatoires ; il y a une seule clé primaire par table. Les autres clés candidates de la table sont alors appelées clés uniques.

Une colonne ou combinaison de colonnes d'une table qui référence une clé candidate d'une autre table (en général la clé primaire) est appelée clé étrangère. Une table peut avoir plusieurs clés étrangères.

_Exemple_



**auteur**

| Field | Type | Null | Key |
|---|---|---|---|
| id | int(11) | NO | PRI |
| nom | varchar(40) | NO | MUL |
| prenom | varchar(40) | NO | |
| mail | varchar(200) | YES | |
| tel_bureau | varchar(10) | YES | |
| tel_portable | varchar(10) | YES | |
| tel_domicile | varchar(10) | YES | |
| mot_de_passe | blob | YES | |
| profil | blob | YES | |

**rubrique**

| Field | Type | Null | Key |
|---|---|---|---|
| id | int(11) | NO | PRI |
| titre | varchar(20) | NO | |
| id_parent | int(11) | YES | |

**auteur_livre**

| Field | Type | Null | Key |
|---|---|---|---|
| id_auteur | int(11) | NO | PRI |
| id_livre | int(11) | NO | PRI |

**rubrique_livre**

| Field | Type | Null | Key |
|---|---|---|---|
| id_rubrique | int(11) | NO | PRI |
| id_livre | int(11) | NO | PRI |

**livre**

| Field | Type | Null | Key |
|---|---|---|---|
| id | int(11) | NO | PRI |
| isbn | varchar(20) | YES | |
| titre | varchar(100) | NO | MUL |
| sous_titre | varchar(100) | YES | |
| nombre_pages | int(11) | YES | |
| annee_parution | year(4) | YES | |
| niveau | enum('Débutant',...) | YES | |
| id_collection | int(11) | YES | |
| id_promotion | int(11) | YES | |
| description | text | YES | |
| couverture | blob | YES | |
| date_maj | timestamp | NO | |

**collection**

| Field | Type | Null | Key |
|---|---|---|---|
| id | int(11) | NO | PRI |
| nom | varchar(25) | NO | UNI |
| prix_ht | decimal(5,2) | YES | |
| frais_ht | decimal(5,2) | YES | |

**promotion**

| Field | Type | Null | Key |
|---|---|---|---|
| id | int(11) | NO | PRI |
| intitule | varchar(40) | NO | |
| date_debut | date | YES | |
| date_fin | date | YES | |
| est_active | tinyint(1) | YES | |

Le schéma ci-dessus présente le modèle de la base de données utilisée dans cet ouvrage. Ce modèle est un modèle simplifié de gestion des livres d'un éditeur.

Ce modèle comporte les tables suivantes :

`auteur`

Auteurs des livres.

`rubrique`

Rubriques permettant le classement des livres dans différentes catégories (base de données, langage de développement, etc.). Les rubriques sont organisées sur deux niveaux : rubrique principale et sous-rubrique. Une sous-rubrique est rattachée à une rubrique parent par l'intermédiaire de la colonne `id_parent`. Pour une rubrique parent, la colonne `id_parent` est vide.

`collection`

Collections de l'éditeur dans lesquelles les livres sont publiés.

`promotion`

Promotions sur les livres.

`livre`

Livres publiés par l'éditeur.

`auteur_livre`

Relation entre les auteurs et les livres : un auteur peut écrire plusieurs livres et un livre peut avoir plusieurs auteurs.

`rubrique_livre`

Relation entre les rubriques et les livres : une rubrique peut contenir plusieurs livres et un livre peut appartenir à plusieurs rubriques.

Dans toutes les tables, à l'exception de `auteur_livre` et `rubrique_livre`, la clé primaire est la colonne `id`. Pour les tables `auteur_livre` et `rubrique_livre`, la clé primaire est la combinaison des deux colonnes, respectivement (`id_auteur,id_livre`) et (`id_rubrique,id_livre`).

Dans l'état actuel du modèle, il y a une clé unique sur la colonne `nom` de la table `collection`. Dans le chapitre Construire une base de données dans MySQL, nous ajouterons des clés uniques sur d'autres tables.

La table `livre` comporte deux clés étrangères : `id_collection` (vers la table `collection`) et `id_promotion` (vers la table `promotion`).

La table `auteur_livre` comporte deux clés étrangères : `id_auteur` (vers la table `auteur`) et `id_livre` (vers la table `livre`).

La table `rubrique_livre` comporte deux clés étrangères : `id_rubrique` (vers la table `rubrique`) et `id_livre` (vers la table `livre`).

La table `rubrique` comporte une clé étrangère : `id_parent` (vers la table `rubrique`).

# Travailler avec MySQL

## 1. Administration du serveur MySQL

Après avoir installé MySQL, nous pouvons administrer le serveur MySQL avec le compte super-utilisateur `root` (rien à voir avec le compte `root` sous Unix ou Linux).

Initialement, le compte `root` n'a pas de mot de passe et il dispose de tous les droits sur toutes les bases de données du serveur MySQL. Par contre, il ne peut se connecter que localement (à partir du serveur lui-même).

Dans le chapitre Construire une base de données dans MySQL, nous verrons comment créer d'autres utilisateurs et leur affecter des droits.

## 2. Interface ligne de commande

L'application cliente `mysql` est un programme interactif qui permet de se connecter à un serveur MySQL et d'exécuter des requêtes sur ce serveur. Cette application se trouve dans le répertoire `bin` de votre installation MySQL.

*Syntaxe*

```
mysql [-h hôte] [-u utilisateur] [-p[mot_de_passe]] [nom_base]
```

`-h` *`hôte`*

Hôte auquel il faut se connecter (machine locale par défaut).

`-u` *`utilisateur`*

Nom d'utilisateur pour la connexion (nom de l'utilisateur courant du système d'exploitation par défaut).

`-p[`*`mot_de_passe`*`]`

Mot de passe pour la connexion (aucun mot de passe par défaut). S'il n'est pas donné sur la ligne de commande, il sera demandé de manière interactive, en saisie masquée. Si le mot de passe est spécifié dans la ligne de commande (ce qui n'est pas conseillé pour la sécurité), il ne doit pas y avoir d'espace après l'option `-p`.

*`nom_base`*

Base sélectionnée au départ (aucune par défaut).

*Exemple*

```
[root@xampp ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 32
Server version: 5.0.45 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Sur cet exemple, la connexion s'effectue localement, sans mot de passe, en tant qu'utilisateur `root`. Comme l'utilisateur courant est `root` au niveau du système d'exploitation, le même résultat peut être obtenu en tapant simplement la commande `mysql` :

```
[root@xampp ~]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

Si un mot de passe est requis et que vous ne souhaitez pas le saisir sur la ligne de commande, vous pouvez utiliser la commande suivante pour vous connecter :

```
[root@xampp ~]# mysql -u root -p
```

```
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

Si un mot de passe est requis et que vous tentez de vous connecter sans mot de passe, vous obtenez une erreur :

```
ERROR 1045 (28000): Access denied for user 'root'@'localhost'
(using password: NO)
```

De même, si vous tentez de vous connecter à partir d'une machine qui n'est pas autorisée, vous obtenez une erreur similaire.

Dans l'interface ligne de commande, vous pouvez saisir soit des commandes du client mysql soit des requêtes SQL.

Les commandes du client mysql sont interprétées directement par le client mysql. Une telle commande doit être écrite sur une seule ligne et l'utilisation du point virgule en fin de commande est superflue. Les commandes du client mysql ne sont pas sensibles à la casse et peuvent être saisies indifféremment en minuscules ou en majuscules.

Les requêtes SQL sont envoyées au serveur pour être exécutées. Une requête SQL peut être écrite sur plusieurs lignes et doit se terminer par un point virgule. Les mots clés du langage SQL ne sont pas sensibles à la casse et peuvent être saisis indifféremment en minuscules ou en majuscules.

Les commandes du client mysql les plus souvent utilisées sont les suivantes :

exit ou quit

Quitte l'application.

use*nom_base*

Utilise une autre base de données.

source*fichier_script*

Exécute un script SQL.

delimiter*caractères*

Modifie le délimiteur utilisé pour terminer une requête SQL.

*Exemple*

```
[root@xampp ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 37
Server version: 5.0.45 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use information_schema
Database changed

mysql> SELECT schema_name FROM schemata;
+--------------------+
| schema_name        |
+--------------------+
| information_schema |
| cdcol              |
| mysql              |
| phpmyadmin         |
| test               |
+--------------------+
5 rows in set (0.01 sec)

mysql> exit
Bye
[root@xampp ~]#
```

Dans la suite de cet ouvrage, nous présenterons la syntaxe de l'ordre SQL SELECT, ainsi que la base de données

information_schema.

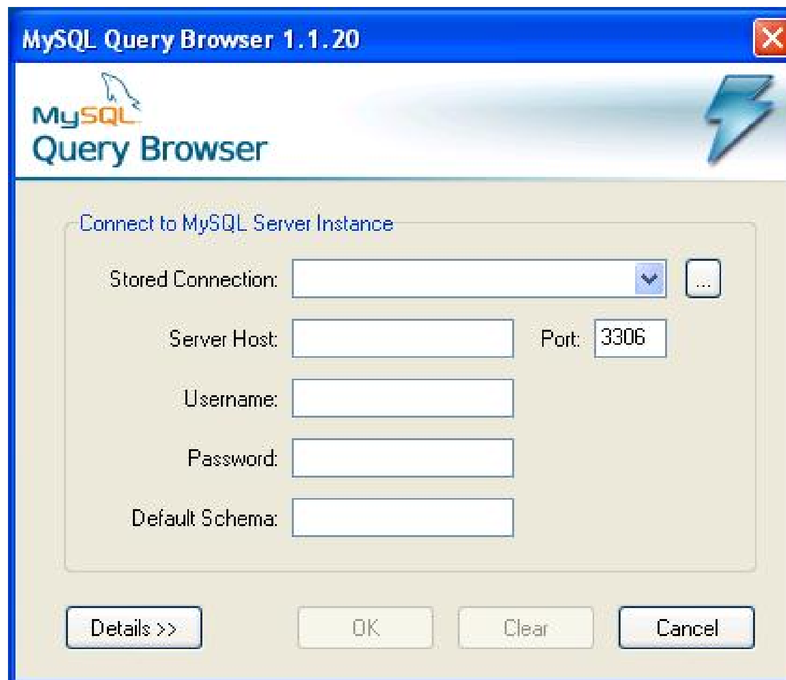
Il existe aussi un ordre SQL USE équivalent à la commande use.

## 3. MySQL Query Browser

MySQL Query Browser est une application graphique développée par la société MySQL AB qui permet d'éditer et d'exécuter des requêtes SQL dans une base de données MySQL.

Vous pouvez vous procurer MySQL Query Browser à l'adresse suivante : http://dev.mysql.com/downloads/. Le produit est disponible sur les plates-formes Windows, Linux et MacOS.

Lorsque vous lancez l'application, une fenêtre de dialogue s'affiche afin de saisir les paramètres de connexion au serveur MySQL :




Dans cette fenêtre de dialogue, vous devez indiquer les informations suivantes :

**Server Host**

Hôte auquel il faut se connecter (obligatoire).

**Username**

Nom d'utilisateur pour la connexion (nom de l'utilisateur courant du système d'exploitation par défaut).

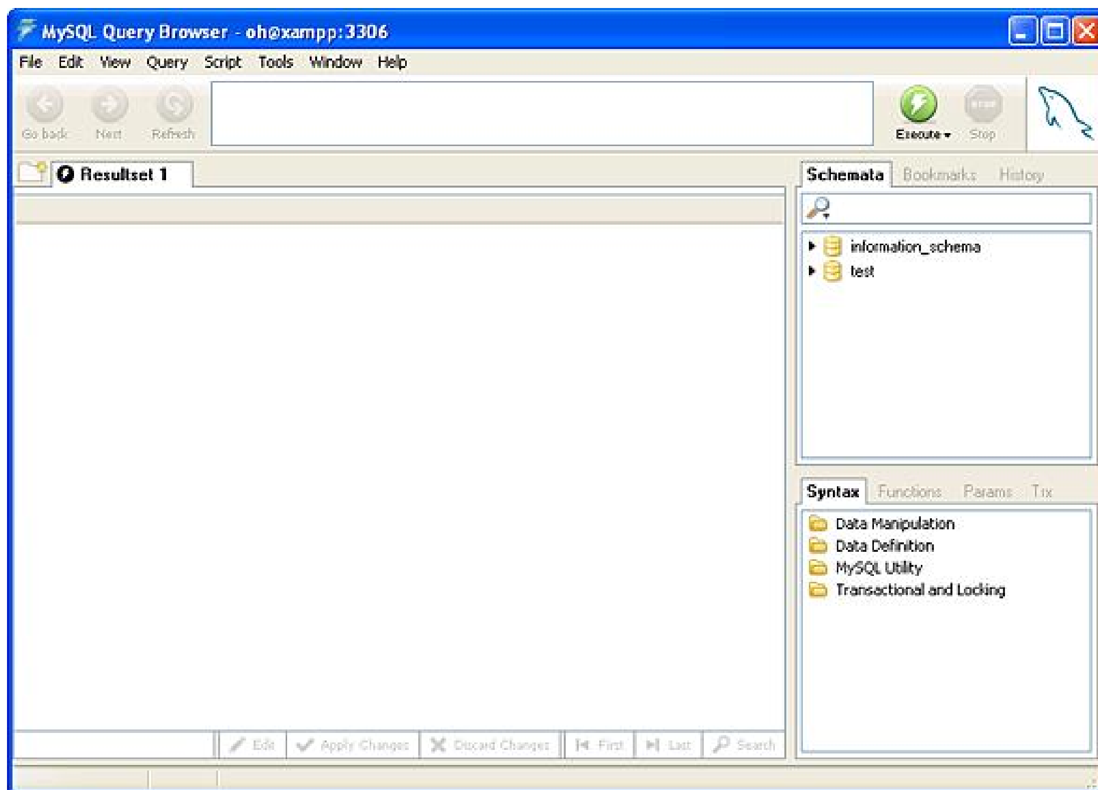
**Password**

Mot de passe pour la connexion (aucun mot de passe par défaut).

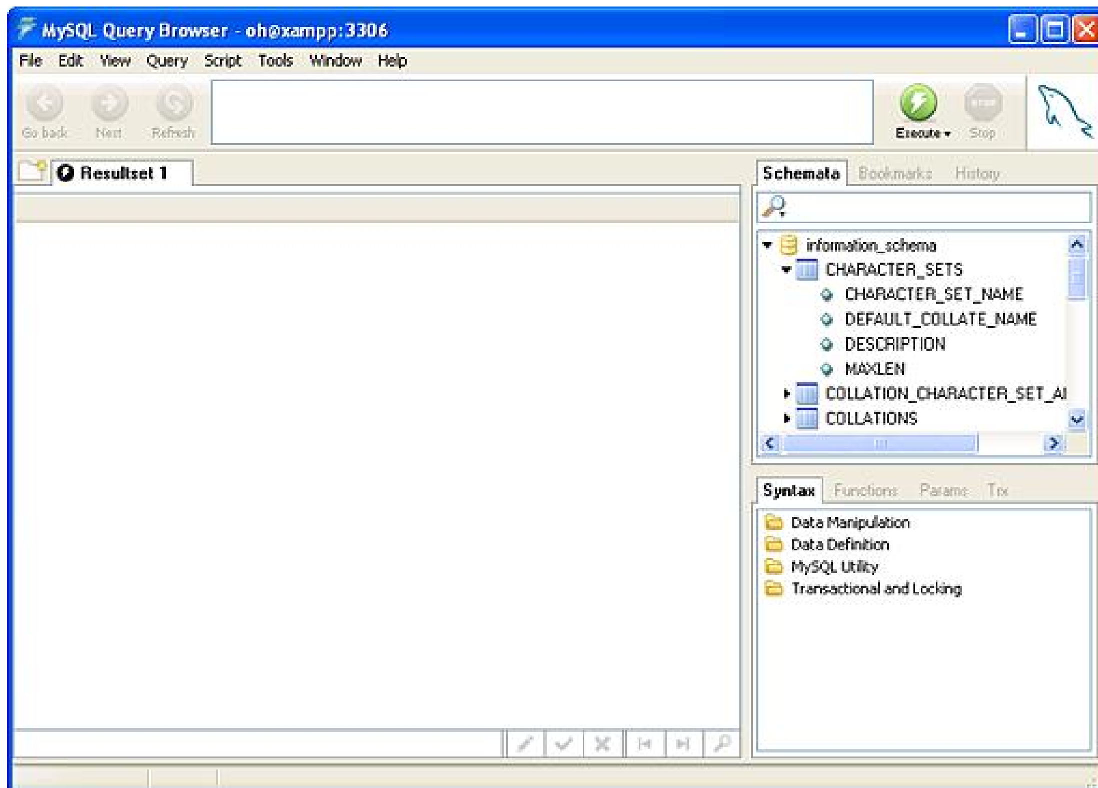
**Default Schema**

Base (ou schéma) sélectionnée au départ (aucune par défaut).

Une fois connecté, la fenêtre suivante s'affiche :

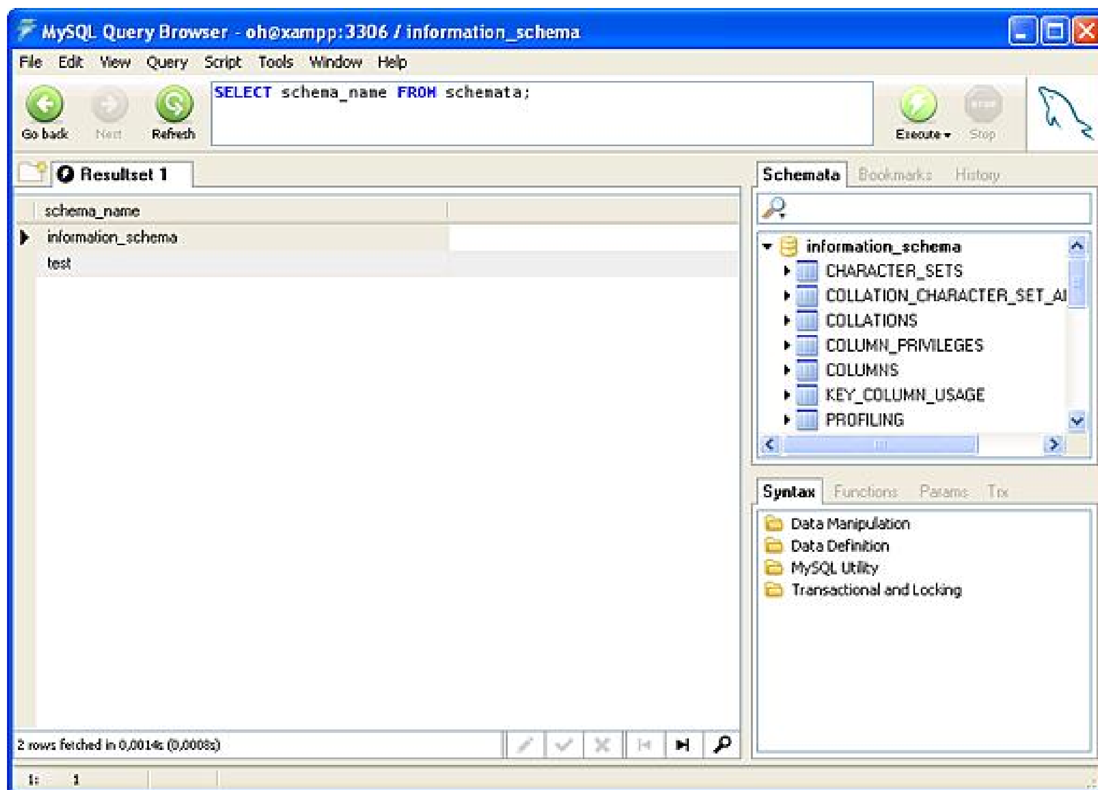



Dans le cadre de droite, l'outil liste les bases de données auxquelles l'utilisateur a accès. En cliquant sur les petits triangles, vous pouvez très facilement afficher la liste des tables stockées dans une base de données, puis la structure d'une table :



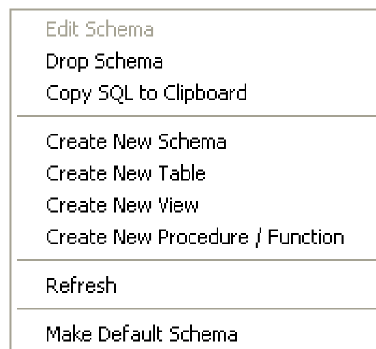

Si vous double cliquez sur le nom d'une base de données, celle-ci devient la base de données courante : son nom est alors affiché en gras.

Dans le cadre supérieur, vous pouvez saisir le texte d'une requête SQL puis cliquer sur le bouton **Execute** pour l'exécuter ; le résultat de la requête s'affiche dans l'onglet **Resultset** :




Si vous double cliquez sur le nom d'une table dans le cadre de droite, une requête permettant d'afficher tout le contenu de la table est automatiquement renseignée dans le cadre supérieur.

Si vous effectuez un clic droit sur un objet dans le cadre de droite, un menu contextuel s'affiche. Ce dernier vous propose différents articles qui vous permettent de créer, modifier ou supprimer les objets (table, vue, programme stocké) :




Dans l'ensemble, cet outil est très convivial et son apprentissage est aisé.

## 4. phpMyAdmin

phpMyAdmin est une application Web développée en PHP qui permet d'administrer un serveur MySQL (sous réserve de disposer d'un compte utilisateur MySQL ayant les droits suffisants).

phpMyAdmin permet de :

- gérer les bases de données du serveur MySQL ;
- gérer les utilisateurs et les droits ;
- gérer les différents objets d'une base de données (tables, colonnes, index, vues, programmes stockés, etc.) ;
- éditer et exécuter des requêtes SQL ;

- charger des fichiers textes dans des tables ;
- exporter ou importer des tables ;
- exporter les données des tables dans différents formats (CSV, XML, PDF, etc) ;
- Etc.

Vous pouvez vous procurer phpMyAdmin à l'adresse suivante : **http://www.phpmyadmin.net/**

phpMyAdmin peut être configuré à l'aide du fichier `config.inc.php`.

Ce fichier de configuration permet notamment de spécifier la méthode d'authentification.

Si vous ne souhaitez pas saisir un nom et un mot de passe lors de l'utilisation de phpMyAdmin, vous pouvez utiliser la méthode d'authentification `config` ; avec cette méthode, le nom de l'utilisateur et le mot de passe sont stockés dans le fichier de configuration. Ce dernier doit contenir les trois lignes suivantes :

```
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root'; // nom de l'utilisateur
$cfg['Servers'][$i]['password'] = 'xhz12A8q0'; // mot de passe
```

Si vous souhaitez sécuriser l'utilisation de phpMyAdmin, vous pouvez utiliser la méthode d'authentification `cookie` ; avec cette méthode, le nom d'utilisateur et le mot de passe sont demandés par phpMyAdmin. Le fichier de configuration doit contenir les deux lignes suivantes :

```
$cfg['blowfish_secret'] = 'abZ123aXiu65';
...
$cfg['Servers'][$i]['auth_type'] = 'cookie';
```

La directive `blowfish_secret` permet de définir une phrase qui sera utilisée pour chiffrer le mot de passe de l'utilisateur.

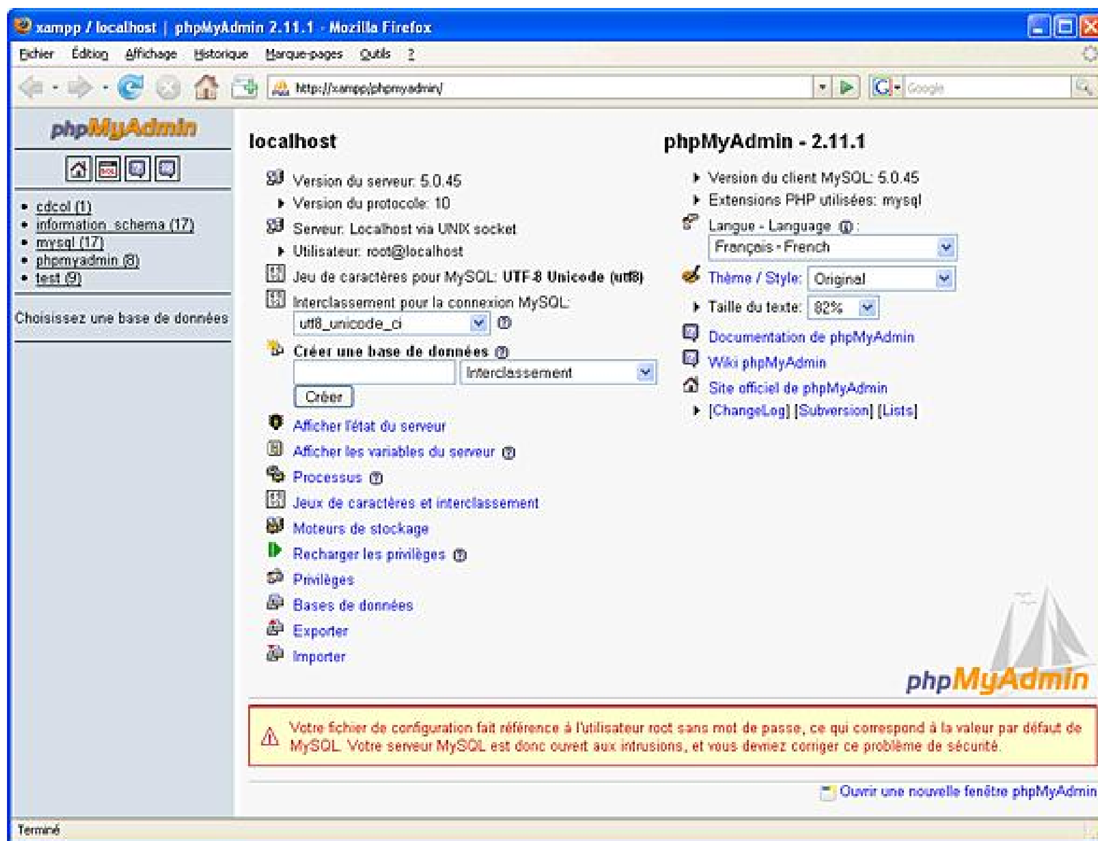
---

Quel que soit le mode d'authentification choisi, le nom d'utilisateur et le mot de passe utilisés pour la connexion à phpMyAdmin doivent être ceux d'un compte MySQL valide. Consultez la documentation de phpMyAdmin pour obtenir plus d'informations sur le fichier de configuration et les différents modes d'authentification.

---

## Page d'accueil

La page d'accueil de phpMyAdmin affiche quelques informations générales sur le serveur :

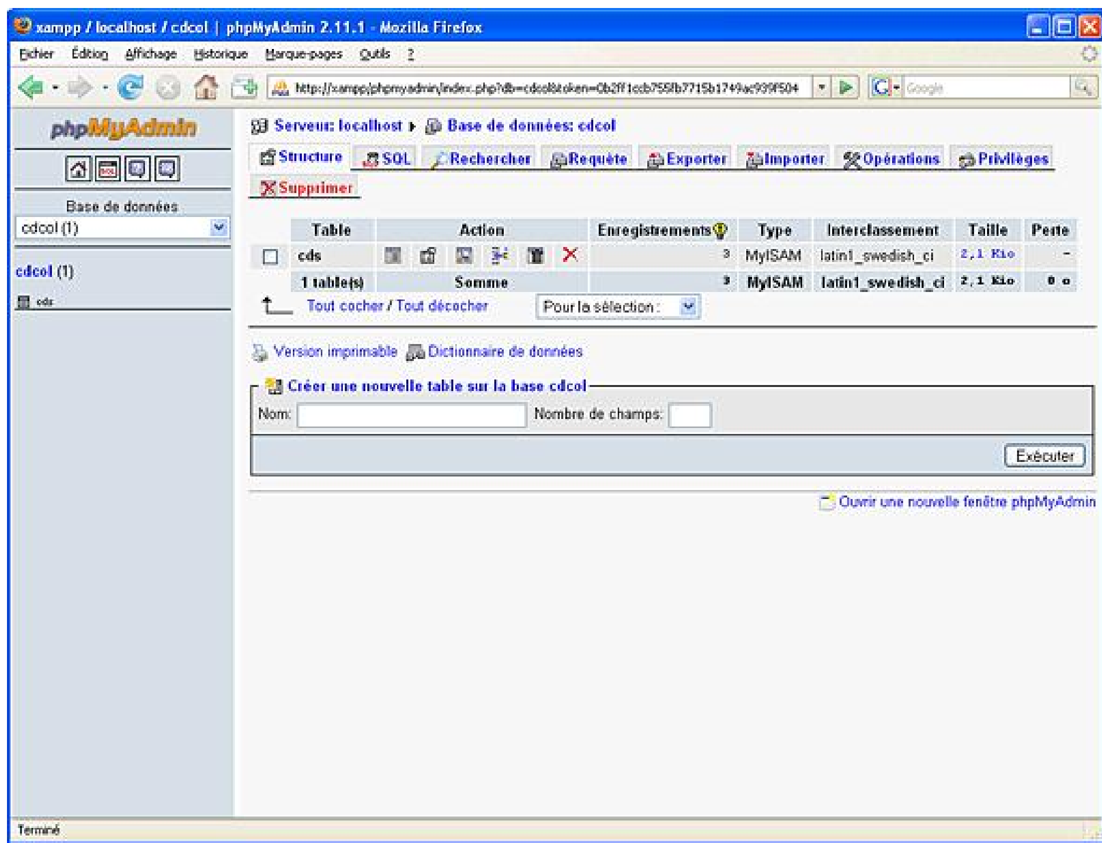



Entre autres actions, cette page d'accueil permet de créer une nouvelle base de données (formulaire **Créer une base de données**).

Dans la partie gauche de la fenêtre, phpMyAdmin affiche la liste des bases de données gérées par le serveur MySQL.

Sur la page d'accueil, vous pouvez cliquer sur le lien correspondant à une base de données afin d'accéder à la page d'administration de cette base de données.

### Page d'administration d'une base de données




La page d'administration d'une base de données propose plusieurs onglets :

**Structure**

Cet onglet affiche la liste des tables et la liste des programmes stockés et propose plusieurs liens et icônes pour gérer ces différents objets. Pour éditer un objet, il suffit de cliquer sur l'icône **Structure** ( ) associée.

**SQL**

Cet onglet permet d'écrire des requêtes SQL sur la base de données actuellement sélectionnée.

**Rechercher**

Cet onglet permet de rechercher des données dans une ou plusieurs tables.

**Requête**

Cet onglet propose un éditeur qui permet de construire des requêtes SQL sans connaître le langage SQL.

**Exporter**

Cet onglet permet d'exporter tout ou partie d'une base de données sous différentes formes.

**Importer**

Cet onglet permet d'exécuter les requêtes contenues dans un fichier.

**Opérations**

Cet onglet permet d'effectuer diverses opérations sur la base de données.

**Privilèges**

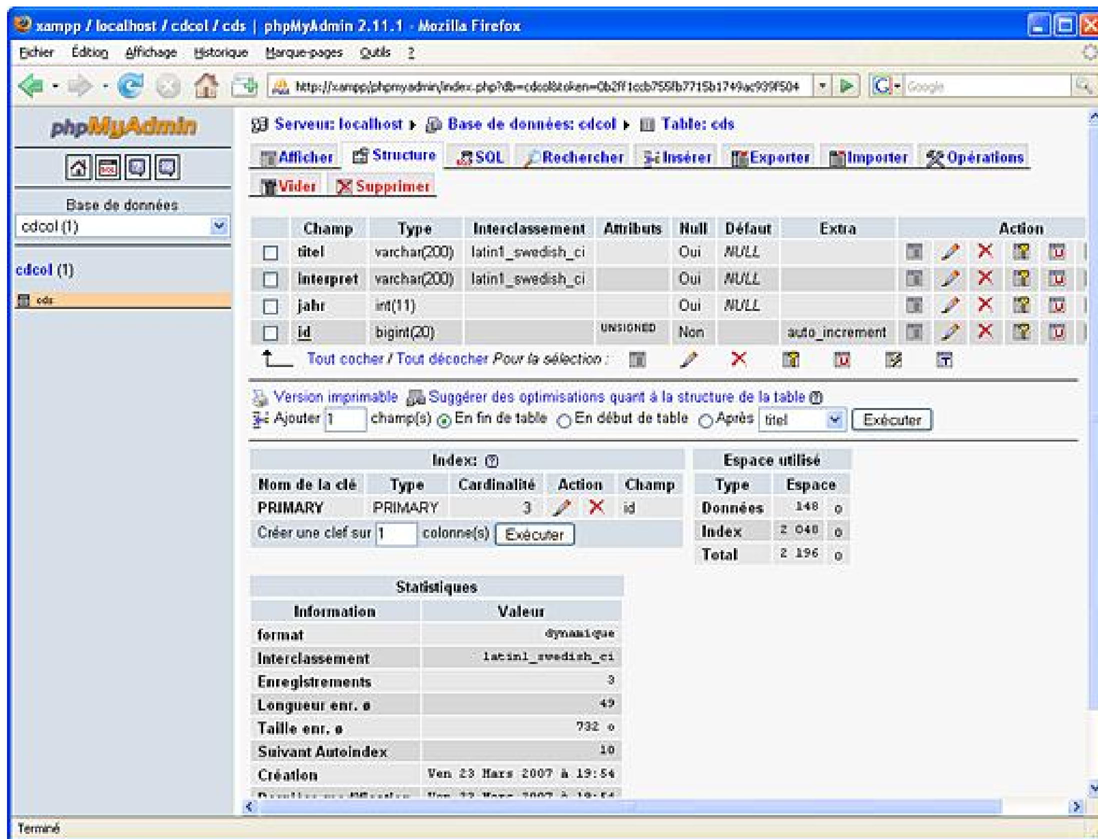
Cet onglet affiche les droits des utilisateurs qui ont accès à la base de données courante.

**Supprimer**

Cet onglet permet de supprimer la base de données courante.

Dans la partie gauche de la fenêtre, phpMyAdmin affiche la liste des tables de la base de données. Vous pouvez cliquer sur le lien correspondant à une table afin de l'éditer.

## Page d'édition d'une table




La page d'édition d'une table propose plusieurs onglets :

### Afficher

Cet onglet affiche les données de la table et permet de les modifier.

### Structure

Cet onglet affiche la structure de la table et propose plusieurs liens et icônes pour modifier cette structure (ajouter ou supprimer des colonnes, des index, etc.). Pour éditer un objet, il suffit de cliquer sur l'icône **Modifier** ( ) associée.

### SQL

Cet onglet permet d'écrire des requêtes SQL sur la table.

### Rechercher

Cet onglet permet de rechercher des données dans la table.

### Insérer

Cet onglet propose un formulaire qui permet d'insérer des nouvelles lignes dans la table.

### Exporter

Cet onglet permet d'exporter la table sous différentes formes.

### Importer

Cet onglet permet d'importer des données dans la table.

**Opérations**

Cet onglet permet d'effectuer diverses opérations sur la table.

**Vider**

Cet onglet permet de vider la table de son contenu sans la supprimer (ordre SQL `TRUNCATE`).

**Supprimer**

Cet onglet permet de supprimer la table.

## 5. Fichier de configuration

Les différents programmes MySQL, dont le serveur `mysqld`, peuvent être configurés par plusieurs options passées en ligne de commande ou spécifiées dans un fichier de configuration.

Sur une plate-forme Linux, le fichier de configuration s'appelle `my.cnf` et se trouve généralement dans le répertoire `/etc`. Sur une plate-forme Windows, le fichier de configuration s'appelle `my.ini` et se trouve généralement dans le répertoire Windows (`c:\windows` par exemple).

Le fichier de configuration comporte plusieurs sections délimitées par des mots entre crochets.

La section `[client]` permet de définir des options pour les programmes clients comme `mysql` : port, mot de passe, etc.

La section `[mysqld]` permet de spécifier des options pour le serveur.

# Installer notre base de données de démonstration

Dans la suite de ce chapitre, nous allons travailler avec une base de données nommée eni.

Cette base de données ENI peut être créée à l'aide du script `creer-base-eni.sql` suivant :

```
-- Création de la base de données.
DROP DATABASE IF EXISTS eni ;
CREATE DATABASE eni;
USE eni;

-- Création de la table RUBRIQUE.
CREATE TABLE rubrique
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  titre VARCHAR(20) NOT NULL,
  id_parent INT
  );
INSERT INTO rubrique
  (id,titre,id_parent)
VALUES
  (1,'Base de données',NULL),
  (2,'Développement',NULL),
  (3,'Internet',NULL),
  (4,'Open Source',NULL)
;
INSERT INTO rubrique
  (titre,id_parent)
VALUES
  ('MySQL',1),
  ('Oracle',1),
  ('Langages',2),
  ('Méthode',2),
  ('HTML - XML',3),
  ('Conception Web',3),
  ('Sécurité',3),
  ('Système',4),
  ('Langages',4),
  ('Base de données',4)
;

-- Création de la table COLLECTION.
CREATE TABLE collection
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(25) NOT NULL UNIQUE,
  prix_ht DECIMAL(5,2) DEFAULT 20,
  frais_ht DECIMAL(5,2)
  );
INSERT INTO collection
  (nom,prix_ht,frais_ht)
VALUES
  ('Ressources Informatiques',24.44,1.5),
  ('TechNote',9.48,NULL),
  ('Les TP Informatiques',25.59,1.5),
  ('Coffret Technique',46.45,2)
;

-- Création de la table AUTEUR.
CREATE TABLE auteur
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nom VARCHAR(40) NOT NULL,
  prenom VARCHAR(40) NOT NULL,
  mail VARCHAR(200),
  tel_bureau VARCHAR(10),
  tel_portable VARCHAR(10),
```

```
  tel_domicile VARCHAR(10),
  mot_de_passe BLOB,
  profil BLOB,
  UNIQUE (nom,prenom)
  );
INSERT INTO auteur
  (nom,prenom,mail,tel_bureau,tel_portable,tel_domicile)
VALUES
  ('HEURTEL','Olivier',NULL,NULL,'0687731346','0102030405'),
  ('THIBAUD','Cyril',NULL,'0203040506',NULL,NULL),
  ('GUERIN','Brice-Arnaud',NULL,NULL,NULL,'0304050607')
;

-- Création de la table PROMOTION.
CREATE TABLE promotion
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  intitule VARCHAR(40) NOT NULL,
  date_debut DATE,
  date_fin DATE,
  est_active BOOLEAN
  );
INSERT INTO promotion
  (intitule,date_debut,date_fin,est_active)
VALUES
  ('-5% sur cet ouvrage',CURDATE(),ADDDATE(CURDATE(),10),TRUE),
  ('Frais de port offerts sur cet ouvrage',NULL,NULL,FALSE),
  ('Un superbe marque page en cadeau',NULL,NULL,FALSE)
;

-- Création de la table LIVRE.
CREATE TABLE livre
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  isbn VARCHAR(20),
  titre VARCHAR(100) NOT NULL,
  sous_titre VARCHAR(100),
  nombre_pages INT,
  annee_parution YEAR(4),
  niveau ENUM('Débutant','Initié','Confirmé','Expert'),
  id_collection INT,
  id_promotion INT,
  description TEXT,
  couverture BLOB,
  date_maj TIMESTAMP
  );
INSERT INTO livre
  (isbn,titre,sous_titre,nombre_pages,annee_parution,niveau,id_collection,
   id_promotion,description,couverture)
VALUES
  ('2-7460-1451-3','PHP 4',
   'Développer un site Web dynamique et interactif',
   447,2001,'Initié',1,1,NULL,NULL),
  ('978-2-7460-3992-6','PHP 5.2',
   'Développer un site Web dynamique et interactif',
   518,2007,'Initié',1,1,NULL,NULL),
  ('2-7460-3104-3','PHP 5',
   'L\'accès aux données (MySQL, Oracle, SQL Server, SQLite...)',
   211,2006,'Expert',2,1,NULL,NULL),
  ('2-7460-2778-X','Oracle 10g',
   'Administration',
   489,2005,'Initié',1,NULL,NULL,NULL),
  ('2-7460-2834-4','Oracle 10g',
   'Installation du serveur sous Windows/Linux - Oracle Net',
   161,2005,'Expert',2,NULL,NULL,NULL),
  ('2-7460-2833-6','Oracle 10g',
   'Sauvegarde et restauration de la base de données avec RMAN',
   174,2005,'Expert',2,NULL,NULL,NULL),
  ('2-7460-2281-8','BusinessObjects 6',
```

```
   NULL,
   470,2004,'Initié',1,NULL,NULL,NULL),
  ('2-7460-3004-7','MySQL 5',
   'Installation, mise en œuvre, administration et programmation',
   468,2006,'Initié',1,NULL,NULL,NULL),
  ('2-7460-2340-7','PHP et MySQL (versions 4 et 5)',
   'Entraînez-vous à créer des applications professionnelles',
   272,2004,'Initié',3,NULL,NULL,NULL),
  ('2-7460-3377-1','MySQL 5 et PHP 5',
   'Maîtrisez les sites web dynamiques',
   972,2006,'Initié',4,2,NULL,NULL)
;

-- Création de la table AUTEUR_LIVRE.
CREATE TABLE auteur_livre
  (
  id_auteur INT,
  id_livre INT,
  PRIMARY KEY (id_auteur,id_livre)
  );
INSERT INTO auteur_livre
  (id_auteur,id_livre)
VALUES
  (1,1),
  (1,2),
  (1,3),
  (1,4),
  (1,5),
  (1,6),
  (1,7),
  (2,8),
  (3,9),
  (1,10),
  (2,10)
;

-- Création de la table RUBRIQUE_LIVRE.
CREATE TABLE rubrique_livre
  (
  id_rubrique INT,
  id_livre INT,
  PRIMARY KEY (id_rubrique,id_livre)
  );
INSERT INTO rubrique_livre
SELECT rub.id,liv.id
FROM livre liv,rubrique rub
WHERE
liv.titre like '%PHP%'
AND rub.titre IN ('Langages','Conception Web')
AND rub.id_parent IS NOT NULL;
INSERT INTO rubrique_livre
SELECT rub.id,liv.id
FROM livre liv,rubrique rub
WHERE
liv.titre like '%MySQL%'
AND rub.titre IN ('MySQL','Base de données')
AND rub.id_parent IS NOT NULL;
INSERT INTO rubrique_livre
SELECT rub.id,liv.id
FROM livre liv,rubrique rub
WHERE
liv.titre like '%Oracle%'
AND rub.titre IN ('Oracle')
AND rub.id_parent IS NOT NULL;

-- Création de la table CATALOGUE.
CREATE TABLE catalogue
  (
  code VARCHAR(10) NOT NULL UNIQUE,
```

```
  titre VARCHAR(100) NOT NULL UNIQUE,
  prix_ttc DECIMAL(5,2) NOT NULL
  );

-- Création de trois programmes stockés
delimiter //
CREATE PROCEDURE ps_creer_collection
  (
  -- Nom de la nouvelle collection.
  IN p_nom VARCHAR(25),
  -- Prix HT de la nouvelle collection.
  IN p_prix_ht DECIMAL(5,2),
  -- Identifiant de la nouvelle collection.
  OUT p_id INT
  )
BEGIN
  /*
  ** Insérer la nouvelle collection et
  ** récupérer l'identifiant affecté.
  */
  INSERT INTO collection (nom,prix_ht)
  VALUES (p_nom,p_prix_ht);
  SET p_id = LAST_INSERT_ID();
END;
//
CREATE PROCEDURE ps_lire_sous_rubriques
  (
  -- Identifiant d'une rubrique (parent).
  IN p_id_parent INT
  )
BEGIN
  /*
  ** Sélectionner les sous-rubriques d'une
  ** rubrique dont l'identifiant est passé
  ** en paramètre.
  */
  SELECT
    titre
  FROM
    rubrique
  WHERE
    id_parent = p_id_parent;
END;
//
CREATE FUNCTION fs_nombre_sous_rubriques
  (
  -- Identifiant d'une rubrique (parent).
  p_id_parent INT
  )
  RETURNS INT
BEGIN
  /*
  ** Compter le nombre de sous-rubriques d'une
  ** rubrique dont l'identifiant est passé
  ** en paramètre.
  */
  DECLARE v_resultat INT;
  SELECT
    COUNT(*)
  INTO
    v_resultat
  FROM
    rubrique
  WHERE
    id_parent = p_id_parent;
  RETURN v_resultat;
END;
//
delimiter ;
```

```
-- Création d'un utilisateur 'eniweb'.
DROP USER eniweb@localhost;
CREATE USER eniweb@localhost IDENTIFIED BY 'web';
GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE ON eni.*
TO eniweb@localhost;

-- Affichage des tables.
SHOW TABLES;
```

Ce script peut être récupéré sur le site de l'éditeur (**www.eni-livres.com**).

Pour créer la base de données `eni` à l'aide du script précédent, vous pouvez procéder de la manière suivante :

- Lancez l'application cliente `mysql` et connectez-vous au serveur en tant qu'utilisateur `root` :

```
[root@xampp ~]# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

- Exécutez le script à l'aide de la commande `source` :

```
mysql> source creer-base-eni.sql
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Database changed
...
...
+----------------+
| Tables_in_eni  |
+----------------+
| auteur         |
| auteur_livre   |
| catalogue      |
| collection     |
| livre          |
| promotion      |
| rubrique       |
| rubrique_livre |
+----------------+
8 rows in set (0.00 sec)
```

Si le script n'est pas stocké dans le répertoire courant, indiquez un chemin complet dans la commande source.

# Apprendre les bases du langage SQL

## 1. Types de données

MySQL propose plusieurs types de données pour la définition des colonnes des tables. Dans le chapitre Construire une base de données dans MySQL, nous verrons comment utiliser ces types de données dans les ordres de création de table.

### a. Types chaîne de caractères

MySQL propose les principaux types suivants pour les chaînes de caractères :

`CHAR[(n)] [BINARY]`

Chaîne de longueur fixe, de `n` caractères (`n` compris entre 0 et `255`, `1` par défaut). Lors du stockage, la chaîne est complétée à droite par des espaces jusqu'à la longueur demandée ; ces espaces sont automatiquement supprimés lorsque la chaîne est lue.

`VARCHAR(n) [BINARY]`

Chaîne de longueur variable, de `n` caractères maximum (`n` compris entre `0` et `65535`, `1` par défaut). Lors du stockage, aucun espace n'est ajouté. Il faut noter que si la chaîne contient des espaces à droite, ceux-ci ne sont pas supprimés lorsque la chaîne est lue.

`TINYTEXT [BINARY]`

Chaîne de longueur variable jusqu'à 255 caractères. équivalent à `VARCHAR(255)`.

`TEXT [BINARY]`

Chaîne de longueur variable jusqu'à $2^{16}$-1 caractères.

`MEDIUMTEXT [BINARY]`

Chaîne de longueur variable jusqu'à $2^{24}$-1 caractères.

`LONGTEXT [BINARY]`

Chaîne de longueur variable jusqu'à $2^{32}$-1 caractères.

`ENUM('valeur'[,...])`

Énumération. Chaîne dont la valeur doit appartenir à une liste de valeurs (ou être `NULL`). Une énumération peut contenir 65 535 valeurs distinctes au maximum.

`SET('valeur'[,...])`

Ensemble. Chaîne qui peut contenir zéro, une ou plusieurs valeurs parmi une liste de valeurs. Un ensemble peut contenir 64 valeurs distinctes au maximum.

Par défaut, les chaînes de caractères ne sont par sensibles à la casse (`'A'` est égal à `'a'`). Le mot clé `BINARY`, dans la définition du type, permet d'avoir une chaîne « binaire » sensible à la casse.

Les types de données `ENUM` et `SET` ne sont pas sensibles à la casse.

---

 Compte tenu de la suppression des espaces lors de la lecture, `'ab '` (espace en fin de chaîne) stocké dans une colonne de type `VARCHAR` n'est pas égal à `'ab '` stocké dans une colonne de type `CHAR`.

---

### b. Types numériques

MySQL propose les principaux types numériques suivants :

TINYINT[(m)]

Entier signé sur 8 bits (-128 à +127).

SMALLINT[(m)]

Entier signé sur 16 bits (-32768 à +32767).

SMALLINT[(m)] UNSIGNED

Entier non signé sur 16 bits (0 à 65535).

MEDIUMINT[(m)]

Entier signé sur 24 bits ($-2^{23}$ à $+2^{23}-1$).

MEDIUMINT[(m)] UNSIGNED

Entier non signé sur 24 bits (0 à $+2^{24}-1$).

INT[(m)]

Entier signé sur 32 bits ($-2^{31}$ à $+2^{31}-1$).

INT[(m)] UNSIGNED

Entier non signé sur 32 bits (0 à $+2^{32}-1$).

BIGINT[(m)]

Entier signé sur 64 bits ($-2^{63}$ à $+2^{63}-1$).

BIGINT[(m)] UNSIGNED

Entier non signé sur 64 bits (0 à $+2^{64}-1$).

FLOAT[(n,d)] [UNSIGNED]

Nombre à virgule flottante en simple précision. n spécifie le nombre de chiffres significatifs et d le nombre de chiffres après la virgule ; si n et d sont omis, MySQL utilise les valeurs maximales permises par la machine. Si le mot clé UNSIGNED est spécifié, les nombres négatifs sont interdits.

DOUBLE[(n,d)] [UNSIGNED]

Nombre à virgule flottante en double précision. n spécifie le nombre de chiffres significatifs et d le nombre de chiffres après la virgule ; si n et d sont omis, MySQL utilise les valeurs maximales permises par la machine. Si le mot clé UNSIGNED est spécifié, les nombres négatifs sont interdits.

FLOAT(p) [UNSIGNED]

Nombre à virgule flottante. P spécifie la précision en bits. Cette précision est utilisée par MySQL pour déterminer le type utilisé : FLOAT entre 0 et 24 et DOUBLE entre 25 et 53.

DECIMAL[(n[,d])] [UNSIGNED]

Nombre à virgule fixe. n spécifie le nombre de chiffres significatifs (10 par défaut, 65 au maximum) et d le nombre de chiffres après la virgule (0 par défaut, 30 au maximum). Si le mot clé UNSIGNED est spécifié, les nombres négatifs sont interdits.

Pour les différents types entiers, le nombre optionnel *m* permet de définir une longueur d'affichage (pas une contrainte sur la plage de valeurs autorisées) ; les valeurs entières dont la longueur est inférieure à cette limite sont affichées complétées à gauche par des espaces. Pour compléter à gauche par des zéros, il est possible d'ajouter le mot clé ZEROFILL à la fin de la définition du type.

INTEGER est un synonyme de INT.

DEC, NUMERIC et FIXED sont des synonymes de DECIMAL.

### c. Type booléen

BOOL, BOOLEAN

Synonymes de TINYINT(1) utilisés pour représenter une valeur booléenne (0 est considéré comme faux ; toute autre valeur est considérée comme vraie).

### d. Types date et heure

MySQL propose les principaux types suivants pour les dates et heures :

DATE

Date comprise entre le 01/01/1000 et 31/12/9999. Format par défaut : YYYY-MM-DD.

DATETIME

Date et heure comprise entre le 01/01/1000 00:00:00 et 31/12/9999 23:59:59. Format par défaut : YYYY-MM-DD HH:MM:SS.

TIMESTAMP

Date et heure comprise entre le 01/01/1970 00:00:01 et 19/01/2038 04:14:08. Format par défaut : YYYY-MM-DD HH:MM:SS.

TIME

Heure comprise entre -838:59:59 et 838:59:59. Format par défaut : HH:MM:SS.

YEAR[(2|4)]

Année sur 2 ou 4 chiffres (4 par défaut), comprise entre 70 (pour 1970) et 69 (pour 2069) pour l'année sur 2 chiffres et entre 1901 et 2155 pour l'année sur 4 chiffres. Format par défaut : YYYY.

---

 Le type de données TIMESTAMP a changé entre la version 4 et la version 5.

---

### e. Types pour les données binaires

MySQL propose les principaux types suivants pour les données binaires (image, son, etc.) :

TINYBLOB

Donnée binaire jusqu'à 255 octets.

BLOB

Donnée binaire jusqu'à $2^{16}$-1 octets.

MEDIUMBLOB

Donnée binaire jusqu'à $2^{24}$-1 octets.

LONGBLOB

Donnée binaire jusqu'à $2^{32}$-1 octets.

Si ces types sont utilisés pour stocker des chaînes de caractères, ces dernières sont sensibles à la casse.

## 2. Nom des objets

Un nom de base de données, de table, de colonne ou d'index est limité à 64 octets.

Pour vous simplifier la tâche, nous vous conseillons de n'utiliser que des caractères non accentués, des chiffres et les caractères `$` et `_` (souligné) pour vos noms d'objets. L'utilisation de tout autre caractère impose en effet de délimiter le nom soit par des apostrophes obliques (`` ` ``) soit par des guillemets (`"`) si la configuration du serveur le permet.

Avec MySQL, les bases de données et les tables correspondent à des dossiers et des fichiers. En conséquence, les noms de bases de données et de tables sont sensibles à la casse si les noms de fichiers le sont au niveau du système d'exploitation. Cela signifie que les noms de bases de données et de tables sont sensibles à la casse sur les plates-formes Unix et Linux et non sensibles à la casse sur les plates-formes Windows.

Les noms de colonnes ne sont pas sensibles à la casse.

Les syntaxes permettant de référencer un nom de table et un nom de colonne sont les suivantes :

```
[nom_base.]nom_table
[[nom_base.]nom_table.]nom_colonne
```

Un nom de table peut être référencé directement si la table est stockée dans la base de données courante. Si la table appartient à une autre base de données, il est nécessaire de préfixer le nom de la table par le nom de la base de données pour y faire référence.

De même, un nom de colonne peut être préfixé par un nom de table (lui-même préfixé par un nom de base de données).

Dans la suite de cet ouvrage, dans les descriptions de syntaxe, les termes `nom_table` et `nom_colonne` désignent respectivement un nom d'une table et un nom de colonne, avec les différentes possibilités de syntaxe présentées ci-dessus.

## 3. Valeurs littérales

### a. Chaîne

Une chaîne littérale est délimitée par des apostrophes : `'ceci est une chaîne'`.

Si la configuration du serveur MySQL le permet, il est aussi possible de délimiter une chaîne littérale par des guillemets. Cette possibilité est déconseillée car elle ne respecte pas la norme ANSI.

À l'intérieur d'une chaîne, certaines séquences de caractères précédées par le caractère d'échappement anti-slash (`\`) ont une signification spéciale :

`\'`

Apostrophe

`\"`

Guillemet

`\n`

Nouvelle ligne

`\r`

Retour chariot

`\t`

Tabulation

`\\`

Anti-slash

Pour intégrer une apostrophe à l'intérieur d'une chaîne littérale délimitée par des apostrophes, il existe deux

possibilités :

- doubler l'apostrophe : `'l''article'` ;
- faire précéder l'apostrophe par le caractère d'échappement anti-slash : `'l\'article'`.

Par défaut, les chaînes de caractères ne sont pas sensibles à la casse, sauf si elles sont définies sous la forme d'une chaîne binaire, ce qui peut être fait en faisant précéder la valeur de l'opérateur `BINARY`. Ainsi, par défaut `'a'` est égal à `'A'`, mais `BINARY'a'` est différent de `'A'`,

### b. Nombre

Un nombre peut être écrit directement, sans délimiteur. Le séparateur décimal est le point (.). La notation exponentielle est autorisée.

*Exemples*

```
123
-10
1.23
1.2e3
```

### c. Date, heure, date/heure

Une date peut être écrite soit sous la forme d'une chaîne au format `'YYYY-MM-DD'`, soit sous la forme d'un nombre au format `YYYYMMDD`.

Une heure peut être écrite soit sous la forme d'une chaîne au format `'HH:MM:SS'`, soit sous la forme d'un nombre au format `HHMMSS`.

Une date/heure peut être écrite soit sous la forme d'une chaîne au format `'YYYY-MM-DD HH:MM:SS'`, soit sous la forme d'un nombre au format `YYYYMMDDHHMMSS`. L'heure est comprise entre 0 et 23.

*Exemples*

```
'2001-12-19'
20031213
'2001-12-19 23:15:00'
20031213031500
```

### d. Booléen

Les deux valeurs booléennes « vrai » et « faux » peuvent être représentées par les constantes `TRUE` (valeur `1`) et `FALSE` (valeur `0`). Le nom de ces constantes n'est pas sensible à la casse.

## 4. Expression

Une expression peut utiliser des valeurs littérales, des noms de colonnes, des variables, les opérateurs arithmétiques habituels (+, -, *, /), des parenthèses et des fonctions (cf. chapitre Utiliser les fonctions MySQL).

*Exemples*

```
prix_ht * ( 1 + taux_tva/100 )
UPPER(titre)
```

## 5. Valeur NULL

La valeur `NULL` signifie l'absence de valeur. `NULL` est différent de `0`, et différent d'une chaîne vide.

Une expression qui contient une valeur `NULL` donne toujours un résultat `NULL` (par exemple, `10 + NULL` est égal à `NULL`).

Les valeurs `NULL` sont souvent source de problème ; dans la suite de cet ouvrage, nous verrons comment éviter ces

problèmes.

# 6. Variables

## a. Variables utilisateurs

MySQL permet d'utiliser des variables utilisateurs spécifiques à la connexion avec la syntaxe `@nom_variable`. Un nom de variable peut contenir des caractères alphanumériques, ainsi que les caractères **_**, **$** et **.**.

Les variables n'ont pas besoin d'être initialisées avant d'être utilisées ; par défaut, elles contiennent la valeur NULL. Une variable utilisateur peut contenir un nombre (entier ou réel) ou une chaîne de caractères.

Les variables peuvent être utilisées dans des expressions.

Pour affecter une valeur à une variable, vous pouvez utiliser la commande SET.

*Syntaxe*

```
SET @nom_variable = expression [,...]
```

Pour afficher la valeur d'une variable, vous pouvez utiliser une syntaxe simplifiée de l'ordre SELECT.

*Syntaxe*

```
SELECT @nom_variable [,...]
```

*Exemple*

```
mysql> SET @taux_tva=19.6;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ROUND(100 * ( 1 + @taux_tva/100 ),2);
+--------------------------------------+
| ROUND(100 * ( 1 + @taux_tva/100 ),2) |
+--------------------------------------+
|                               119.60 |
+--------------------------------------+
1 row in set (0.00 sec)
```

## b. Variables systèmes

MySQL fournit un grand nombre de variables qui permettent d'obtenir des informations sur le fonctionnement du serveur. Dans la plupart des cas, la valeur de ces variables peut être modifiée pour changer le fonctionnement du serveur.

Il existe deux types de variables :

- Les variables globales qui affectent l'ensemble du serveur. Ces variables globales sont initialisées au démarrage du serveur, à partir du fichier de configuration et des arguments passés en ligne de commande au serveur MySQL.
- Les variables de session qui affectent les connexions. Ces variables de session sont initialisées au moment de la connexion à partir des valeurs des variables globales correspondantes.

La commande SET permet de modifier la valeur d'une variable globale ou de session.

Syntaxes

```
SET GLOBAL nom_variable = valeur
SET @@global.nom_variable = valeur
SET [SESSION] nom_variable = valeur
SET [@@[session.]]nom_variable = valeur
```

Les deux premières syntaxes permettent de modifier une variable globale et les deux dernières une variable de session (SESSION et session sont optionnels). LOCAL est synonyme de SESSION.

La modification d'une variable globale n'affecte pas les sessions déjà connectées (pas même la session qui effectue la modification).

Pour afficher la valeur d'une variable globale ou de session, vous pouvez utiliser une syntaxe simplifiée de l'ordre SELECT.

*Syntaxes*

```
SELECT @@global.nom_variable
SELECT @@[session.]nom_variable
```

*Exemple*

```
mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> SET @@lc_time_names=fr_FR;
Query OK, 0 rows affected (0.00 sec)
```

## 7. Commentaires

MySQL supporte trois types de commentaires :

- depuis le caractère '# '(dièse) jusqu'à la fin de la ligne ;
- depuis la séquence '-- ' (deux tirets suivis d'un espace) jusqu'à la fin de la ligne ;
- entre les séquences '/*' et '*/', ces deux séquences pouvant être sur des lignes différentes.

*Exemple*

```
/*
   Exemple de commentaire écrit sur
   plusieurs lignes
*/
SELECT *
FROM collection
ORDER BY rand(); -- tri aléatoire ! (commentaire mono-ligne)
```

# Exécuter des requêtes SQL simples

## 1. Le mode SQL du serveur

Le serveur MySQL peut fonctionner selon différents modes SQL. Le mode SQL définit quelle syntaxe SQL peut être utilisée et quelles vérifications le serveur doit faire, notamment lors des mises à jour.

Le mode SQL utilisé au démarrage est défini par la valeur de l'option `sql-mode` (sur la ligne de commande du démon MySQL ou dans le fichier de configuration).

Depuis la version 4.1, vous pouvez modifier le mode SQL après le démarrage grâce à la variable système`sql_mode`. Le mode SQL peut être modifié au niveau global ou au niveau session (chaque connexion peut utiliser un mode différent adapté à ces besoins).

Le mode SQL est défini par une liste de mots clés séparés par des virgules. La valeur par défaut est vide (pas de mode configuré).

Avec le mode par défaut (pas de mode), le serveur MySQL évite de générer des erreurs lors des mises à jour et n'hésite pas à modifier des valeurs pour permettre leur affectation à une colonne ; une simple alerte est émise.

Pour modifier ce comportement, les modes `STRICT_TRANS_TABLES` ou `STRICT_ALL_TABLES` peuvent être utilisés : c'est le mode SQL "strict". Dans ce mode, les valeurs invalides sont rejetées et une erreur est générée. Le mode `STRICT_ALL_TABLES` s'applique à tous les moteurs de stockage ; le mode `STRICT_TRANS_TABLES` s'applique uniquement aux moteurs de stockage transactionnel.

Nous reviendrons plus en détail sur ce sujet dans la suite de ce chapitre.

---

La notion de moteur de stockage est présentée dans le chapitre Construire une base de données dans MySQL - Gérer les tables.

---

*Exemple d'affichage et de modification du mode SQL*

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> SET @@sql_mode=STRICT_ALL_TABLES;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+-------------------+
| @@sql_mode        |
+-------------------+
| STRICT_ALL_TABLES |
+-------------------+
1 row in set (0.00 sec)
```

---

Il existe d'autres modes SQL (`ANSI`, `TRADITIONAL`, etc.). Pour en savoir plus, consultez la documentation.

---

## 2. Obtenir des informations

L'instruction SQL `SHOW` offre différentes formes qui permettent d'obtenir des informations sur le serveur MySQL, les bases de données qu'il gère et le contenu de ces bases de données (les tables, les colonnes, etc.).

Dans cette section, nous présentons les formes les plus utilisées de la commande `SHOW` permettant d'afficher des informations sur les bases de données gérées par le serveur, les tables stockées dans une base de données et la structure d'une table :

```
SHOW DATABASES
```

`SHOW SCHEMAS`

Liste des bases de données gérées par le serveur mySQL.

`SHOW TABLES`

Liste des tables stockées dans la base de données courante.

`SHOW COLUMNS FROM nom_table`

`{ DESCRIBE | DESC } nom_table`

Liste des colonnes d'une table.

*Exemple*

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| cdcol              |
| eni                |
| mysql              |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.00 sec)

mysql> USE eni;
Database changed
mysql> SHOW TABLES;
+----------------+
| Tables_in_eni  |
+----------------+
| auteur         |
| auteur_livre   |
| catalogue      |
| collection     |
| livre          |
| promotion      |
| rubrique       |
| rubrique_livre |
+----------------+
6 rows in set (0.00 sec)

mysql> DESC collection;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom      | varchar(25)  | NO   | UNI |         |                |
| prix_ht  | decimal(5,2) | YES  |     | 20.00   |                |
| frais_ht | decimal(5,2) | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql>
```

Dans le chapitre Construire une base de données dans MySQL, nous verrons d'autres variantes de la commande `SHOW` qui permettent de récupérer des informations sur les vues, les procédures et les triggers.

## 3. Afficher les erreurs et les alertes

La commande `SHOW WARNINGS` permet d'afficher les alertes ou les erreurs générées par la dernière commande. La

commande SHOW ERRORS permet d'afficher uniquement les erreurs générées par la dernière commande.

Dans l'application cliente mysql, les erreurs générées par la dernière commande sont automatiquement affichées. Par contre, pour les alertes, l'application indique uniquement le nombre d'alertes générées mais ne les affiche pas ; la commande SHOW WARNINGS est alors très utile.

*Exemple*

```
mysql> INSERT INTO rubrique(titre)
    -> VALUES('Messagerie et travail de groupe');
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+------------------------------------------+
| Level   | Code | Message                                  |
+---------+------+------------------------------------------+
| Warning | 1265 | Data truncated for column 'titre' at row 1 |
+---------+------+------------------------------------------+
1 row in set (0.00 sec)
```

## 4. Lire les données

### a. Syntaxe de base de l'ordre SELECT

L'ordre SQL SELECT permet de lire les données.

La syntaxe la plus simple pour l'ordre SELECT est la suivante :

```
SELECT [DISTINCT] expression[,...] | *
FROM nom_table
```

La clause FROM indique dans quelle table lire les données. Nous verrons plus loin comment lire les données dans plusieurs tables.

La clause SELECT contient une liste d'expressions séparées par une virgule. Une expression peut être une colonne d'une table, une valeur littérale ou une expression calculée utilisant éventuellement une ou plusieurs colonnes d'une table. À la place d'une liste d'expressions, il est possible d'utiliser un astérisque pour indiquer que toutes les colonnes sont sélectionnées.

Avec la syntaxe suivante, toutes les lignes de la table sont retournées. Nous verrons plus loin comment restreindre le résultat d'une requête.

*Exemples*

```
mysql> SELECT * FROM collection;
+----+--------------------------+---------+----------+
| id | nom                      | prix_ht | frais_ht |
+----+--------------------------+---------+----------+
|  1 | Ressources Informatiques |   24.44 |     1.50 |
|  2 | TechNote                 |    9.48 |     NULL |
|  3 | Les TP Informatiques     |   25.59 |     1.50 |
|  4 | Coffret Technique        |   46.45 |     2.00 |
+----+--------------------------+---------+----------+
4 rows in set (0.00 sec)

mysql> SELECT id,nom FROM collection;
+----+--------------------------+
| id | nom                      |
+----+--------------------------+
|  1 | Ressources Informatiques |
|  2 | TechNote                 |
|  3 | Les TP Informatiques     |
|  4 | Coffret Technique        |
+----+--------------------------+
4 rows in set (0.00 sec)

mysql> SELECT nom,prix_ht+ROUND(prix_ht*5.5/100,2) FROM collection;
+--------------------------+----------------------------------+
```

```
| nom                       | prix_ht+ROUND(prix_ht*5.5/100,2) |
+---------------------------+----------------------------------+
| Ressources Informatiques  |                            25.78 |
| TechNote                  |                            10.00 |
| Les TP Informatiques      |                            27.00 |
| Coffret Technique         |                            49.00 |
+---------------------------+----------------------------------+
4 rows in set (0.00 sec)
```

Sur le dernier exemple, `ROUND(prix_ht*(1+5.5/100),2)` est une expression calculée sur la colonne `prix_ht`.

Le mot clé `DISTINCT` permet d'obtenir une seule occurrence de chaque ensemble de lignes dupliquées.

*Exemple*

```
mysql> SELECT annee_parution FROM livre;
+----------------+
| annee_parution |
+----------------+
|           2001 |
|           2007 |
|           2006 |
|           2005 |
|           2005 |
|           2005 |
|           2004 |
|           2006 |
|           2004 |
|           2006 |
+----------------+
10 rows in set (0.00 sec)

mysql> SELECT DISTINCT annee_parution FROM livre;
+----------------+
| annee_parution |
+----------------+
|           2001 |
|           2007 |
|           2006 |
|           2005 |
|           2004 |
+----------------+
5 rows in set (0.00 sec)

mysql> SELECT annee_parution,niveau FROM livre;
+----------------+--------+
| annee_parution | niveau |
+----------------+--------+
|           2001 | Initié |
|           2007 | Initié |
|           2006 | Expert |
|           2005 | Initié |
|           2005 | Expert |
|           2005 | Expert |
|           2004 | Initié |
|           2006 | Initié |
|           2004 | Initié |
|           2006 | Initié |
+----------------+--------+
10 rows in set (0.00 sec)

mysql> SELECT DISTINCT annee_parution,niveau FROM livre;
+----------------+--------+
| annee_parution | niveau |
+----------------+--------+
|           2001 | Initié |
|           2007 | Initié |
|           2006 | Expert |
|           2005 | Initié |
|           2005 | Expert |
```

```
|           2004 | Initié |
|           2006 | Initié |
+----------------+--------+
7 rows in set (0.00 sec)
```

L'ordre SELECT peut aussi être utilisé pour lire des expressions calculées, ou des variables, sans référence à une table. Dans ce cas, la clause FROM est absente.

*Exemple*

```
mysql> SELECT 1+2+3;
+-------+
| 1+2+3 |
+-------+
|     6 |
+-------+
1 row in set (0.00 sec)
```

Depuis la version 4.1, vous pouvez spécifier dual comme nom de table dans la clause FROM, dans le cas où aucune table n'est référencée. Cette possibilité permet d'être compatible avec les serveurs de base de données qui ne supportent pas le SELECT sans clause FROM.

*Exemple*

```
mysql> SELECT 1+2+3 FROM dual;
+-------+
| 1+2+3 |
+-------+
|     6 |
+-------+
1 row in set (0.00 sec)
```

Dans la clause SELECT une expression peut être suivie d'un alias qui permet de nommer l'expression.

*Syntaxe*

```
expression [AS] alias
```

Le mot clé AS est optionnel.

*Exemple*

```
mysql> SELECT nom,prix_ht+ROUND(prix_ht*5.5/100,2) prix_ttc
    -> FROM collection;
+--------------------------+----------+
| nom                      | prix_ttc |
+--------------------------+----------+
| Ressources Informatiques |    25.78 |
| TechNote                 |    10.00 |
| Les TP Informatiques     |    27.00 |
| Coffret Technique        |    49.00 |
+--------------------------+----------+
4 rows in set (0.00 sec)
```

Un alias peut être appliqué à une simple colonne, ce qui a pour effet de renommer la colonne dans l'interrogation uniquement (pas dans la table).

 Si l'alias comporte des espaces ou tout autre caractère spécial, il doit être saisi entre guillemets.

De même, dans la clause FROM, un nom de table peut être suivi d'un alias qui permet de renommer la table dans l'interrogation uniquement (pas dans la base de données).

*Syntaxe*

```
nom_table [AS] alias
```

Le mot clé AS est optionnel.

Un alias de table peut être utilisé à la place du nom de la table pour préfixer une colonne.

*Exemple*

```
mysql> SELECT col.nom FROM collection col;
+-------------------------+
| nom                     |
+-------------------------+
| Ressources Informatiques |
| TechNote                |
| Les TP Informatiques    |
| Coffret Technique       |
+-------------------------+
4 rows in set (0.00 sec)
```

Cette possibilité est particulièrement intéressante dans les interrogations portant sur plusieurs tables.

En effet, dans ce genre d'interrogation, il est conseillé de préfixer le nom des colonnes par le nom de la table (ou par un alias de table) pour faciliter la lecture de la requête (on voit toute de suite à quelle table appartient la colonne). Le préfixe est par ailleurs obligatoire lorsque deux tables ont une colonne portant le même nom et que ce nom est présent dans la requête ; sans préfixe la requête est ambiguë et provoque une erreur.

*Exemple*

```
mysql> SELECT id FROM livre,collection;
ERROR 1052 (23000): Column 'id' in field list is ambiguous
```

Utiliser un alias de table à la place du nom de la table pour préfixer les colonnes est intéressant lorsque les tables portent des noms "longs" ; des alias de 3 ou 4 caractères bien choisis (suffisamment significatifs) peuvent avantageusement les remplacer.

---

Pour vous simplifier la tâche, nous vous conseillons de n'utiliser que des caractères non accentués, des chiffres et les caractères $ et _ (souligné) pour vos alias de table. L'utilisation de tout autre caractère imposera en effet de délimiter l'alias soit par des apostrophes obliques (`) soit par des guillemets (") si la configuration du serveur le permet.

---

### b. Restreindre le résultat : clause WHERE

Une clause `WHERE` peut être ajoutée à la requête `SELECT` pour restreindre le résultat.

*Syntaxe*

```
SELECT [DISTINCT] expression[,...] | *
FROM nom_tableWHERE conditions
```

Seules les lignes vérifiant les conditions de la clause `WHERE` sont retournées.

La syntaxe générale d'une condition est la suivante :

```
expression1 opérateur expression2
```

`expression1` et `expression2` sont des expressions du type de celles qui peuvent figurer dans la clause `SELECT` : colonne, valeur littérale ou expression calculée.

Une condition retourne `TRUE`, `FALSE` ou `NULL`.

Les opérateurs les plus souvent utilisés sont les suivants :

=

Égalité

>

Strictement supérieur

>=

Supérieur ou égal

<

Strictement inférieur

<=

Inférieur ou égal

<> ou !=

Différent

BETWEEN*min*AND*max*

Supérieur ou égal à *min* et inférieur ou égal à *max*

IN (valeur,...)

Égalité avec n'importe quel élément d'une liste

IS NULL, IS NOT NULL

Teste si une expression est NULL ou non

LIKE

Correspondance par rapport à un modèle

Les opérateurs BETWEEN, IN et LIKE peuvent être inversés par le mot clé NOT (NOT BETWEEN, NOT IN et NOT LIKE).

*Exemples*

```
mysql> SELECT nom FROM collection WHERE id = 1;
+-------------------------+
| nom                     |
+-------------------------+
| Ressources Informatiques |
+-------------------------+
1 row in set (0.01 sec)

mysql> SELECT id,nom FROM collection WHERE id <> 1;
+----+---------------------+
| id | nom                 |
+----+---------------------+
|  2 | TechNote            |
|  3 | Les TP Informatiques |
|  4 | Coffret Technique   |
+----+---------------------+
3 rows in set (0.00 sec)1 row in set (0.00 sec)

mysql> SELECT id,nom FROM collection WHERE id IN (1,2);
+----+-------------------------+
| id | nom                     |
+----+-------------------------+
|  1 | Ressources Informatiques |
|  2 | TechNote                |
+----+-------------------------+
2 rows in set (0.01 sec)

mysql> SELECT nom,prix_ht FROM collection WHERE prix_ht < 25;
+-------------------------+---------+
| nom                     | prix_ht |
+-------------------------+---------+
| Ressources Informatiques |   24.44 |
| TechNote                |    9.48 |
+-------------------------+---------+
```

```
2 rows in set (0.00 sec)

mysql> SELECT nom,prix_ht FROM collection
    -> WHERE prix_ht BETWEEN 10 AND 25;
+-------------------------+---------+
| nom                     | prix_ht |
+-------------------------+---------+
| Ressources Informatiques |   24.44 |
+-------------------------+---------+
1 row in set (0.00 sec)

mysql> SELECT prix_ht FROM collection WHERE nom = 'TECHNOTE';
+---------+
| prix_ht |
+---------+
|    9.48 |
+---------+
1 row in set (0.00 sec)

mysql> -- Recherche sensible  la casse
mysql> SELECT prix_ht FROM collection WHERE nom = BINARY 'TECHNOTE';
Empty set (0.00 sec)

mysql> -- Recherche sur un couple de colonnes
mysql> SELECT sous_titre FROM livre
    -> WHERE (titre,annee_parution) = ('Oracle 10g',2005);
+---------------------------------------------------------+
| sous_titre                                              |
+---------------------------------------------------------+
| Administration                                          |
| Installation du serveur sous Windows/Linux - Oracle Net    |
| Sauvegarde et restauration de la base de données avec RMAN |
+---------------------------------------------------------+
3 rows in set (0.00 sec)
```

Les colonnes de type DATETIME ou TIMESTAMP sont susceptibles de stocker une composante horaire. Cette composante horaire peut poser des problèmes pour la recherche uniquement sur la partie date. Une première solution consiste à effectuer une recherche bornée ; une deuxième solution consiste à appliquer la fonction DATE à la colonne pour éliminer la composante horaire (cf. chapitre Utiliser les fonctions MySQL - Fonctions dates).

*Exemple*

```
mysql> SELECT titre FROM livre WHERE date_maj = '2008-01-19';
Empty set (0.01 sec)

mysql> SELECT titre FROM livre WHERE DATE(date_maj) = '2008-01-19';
+-------------------------------+
| titre                         |
+-------------------------------+
| PHP 5.2                       |
+-------------------------------+
1 row in set (0.01 sec)

mysql> SELECT titre FROM livre
    -> WHERE date_maj BETWEEN
    ->           '2008-01-19 00:00:00' AND '2008-01-19 23:59:59';
+-------------------------------+
| titre                         |
+-------------------------------+
| PHP 5.2                       |
+-------------------------------+
1 row in set (0.01 sec)
```

Pour tester si une expression est NULL (ou non NULL), il ne faut pas utiliser l'opérateur = (ou !=) mais l'opérateur IS NULL (ou IS NOT NULL). Utiliser les opérateurs = ou != ne génère pas d'erreur, mais l'interrogation ne retourne aucun résultat.

*Exemple*

```
mysql> SELECT titre FROM livre WHERE sous_titre = NULL;
Empty set (0.00 sec)

mysql> SELECT titre FROM livre WHERE sous_titre IS NULL;
+-------------------+
| titre             |
+-------------------+
| BusinessObjects 6 |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT DISTINCT titre FROM livre
    -> WHERE sous_titre IS NOT NULL;
+---------------------------------+
| titre                           |
+---------------------------------+
| PHP 4                           |
| PHP 5.2                         |
| PHP 5                           |
| Oracle 10g                      |
| MySQL 5                         |
| PHP et MySQL (versions 4 et 5)  |
| MySQL 5 et PHP 5                |
+---------------------------------+
7 rows in set (0.00 sec)
```

Avec l'opérateur LIKE, il est possible d'utiliser deux caractères "joker" :

%

Remplace un nombre quelconque de caractères (y compris aucun).

_ (souligné)

Remplace exactement un caractère.

Pour rechercher un de ces deux caractères, vous devez le faire précéder du caractère d'échappement anti-slash dans le modèle. Si vous souhaitez utiliser un autre caractère d'échappement, vous pouvez ajouter la clause ESCAPE 'c', c étant égal au caractère d'échappement utilisé.

*Exemple*

```
mysql> SELECT DISTINCT titre FROM livre WHERE titre LIKE 'PHP%';
+---------------------------------+
| titre                           |
+---------------------------------+
| PHP 4                           |
| PHP 5.2                         |
| PHP 5                           |
| PHP et MySQL (versions 4 et 5)  |
+---------------------------------+
4 rows in set (0.00 sec)

mysql> SELECT DISTINCT titre FROM livre
    -> WHERE titre LIKE '%MySQL%PHP%';
+------------------+
| titre            |
+------------------+
| MySQL 5 et PHP 5 |
+------------------+
1 row in set (0.00 sec)
```

Plusieurs conditions simples peuvent être combinées à l'aide des opérateurs logiques habituels OR, AND et NOT. La précédente par défaut de ces opérateurs est NOT, AND et OR (du plus prioritaire au moins prioritaire). Pour évaluer les conditions dans un ordre différent, il faut utiliser des parenthèses.

*Exemple*

```
mysql> SELECT titre FROM livre
```

```
    -> WHERE annee_parution = 2005 AND id_collection = 1;
+------------+
| titre      |
+------------+
| Oracle 10g |
+------------+
1 row in set (0.00 sec)

mysql> SELECT titre,annee_parution FROM livre
    -> WHERE (annee_parution = 2005 OR annee_parution = 2006)
    ->       AND id_collection = 1;
+------------+----------------+
| titre      | annee_parution |
+------------+----------------+
| Oracle 10g |           2005 |
| MySQL 5    |           2006 |
+------------+----------------+
2 rows in set (0.01 sec)
```

### c. Trier le résultat : clause ORDER BY

Par défaut, le résultat d'une interrogation est retourné dans un ordre indéterminé.

Une clause ORDER BY peut être ajoutée à la requête SELECT pour trier le résultat.

*Syntaxe*

```
SELECT [DISTINCT] expression[,...] | *
FROM nom_table
[WHERE conditions]
ORDER BY expression [ASC | DESC][,...]
```

Si la requête comporte une clause WHERE, la clause ORDER BY doit figurer après.

Le tri peut être croissant (ASC) ou décroissant (DESC) ; il est croissant par défaut. La clause ORDER BY peut contenir plusieurs expressions de tri séparées par des virgules ; l'ordre des expressions détermine l'ordre de priorité des niveaux de tri.

expression peut être :

- une colonne ;
- une expression basée sur des colonnes ;
- un alias de colonne ;
- un numéro correspondant à la position d'une expression de la clause SELECT (syntaxe déconseillée et obsolète).

*Exemples*

```
mysql> SELECT nom,prix_ht+ROUND(prix_ht*5.5/100,2)
    -> FROM collection
    -> ORDER BY nom;
+--------------------------+---------------------------------+
| nom                      | prix_ht+ROUND(prix_ht*5.5/100,2) |
+--------------------------+---------------------------------+
| Coffret Technique        |                           49.00 |
| Les TP Informatiques     |                           27.00 |
| Ressources Informatiques |                           25.78 |
| TechNote                 |                           10.00 |
+--------------------------+---------------------------------+
4 rows in set (0.00 sec)

mysql> SELECT nom,prix_ht+ROUND(prix_ht*5.5/100,2)
    -> FROM collection
    -> ORDER BY prix_ht+ROUND(prix_ht*5.5/100,2);
```

```
+--------------------------+---------------------------------+
| nom                      | prix_ht+ROUND(prix_ht*5.5/100,2) |
+--------------------------+---------------------------------+
| TechNote                 |                           10.00 |
| Ressources Informatiques |                           25.78 |
| Les TP Informatiques     |                           27.00 |
| Coffret Technique        |                           49.00 |
+--------------------------+---------------------------------+
4 rows in set (0.00 sec)

mysql> SELECT nom,prix_ht+ROUND(prix_ht*5.5/100,2) prix_ttc
    -> FROM collection
    -> ORDER BY prix_ttc DESC;
+--------------------------+----------+
| nom                      | prix_ttc |
+--------------------------+----------+
| Coffret Technique        |    49.00 |
| Les TP Informatiques     |    27.00 |
| Ressources Informatiques |    25.78 |
| TechNote                 |    10.00 |
+--------------------------+----------+
4 rows in set (0.00 sec)

mysql> SELECT id_parent,titre
    -> FROM rubrique
    -> ORDER BY id_parent,titre;
+-----------+---------------------+
| id_parent | titre               |
+-----------+---------------------+
|      NULL | Base de données     |
|      NULL | Développement       |
|      NULL | Internet            |
|      NULL | Messagerie et travai |
|      NULL | Open Source         |
|         1 | MySQL               |
|         1 | Oracle              |
|         2 | Langages            |
|         2 | Méthode             |
|         3 | Conception Web      |
|         3 | HTML - XML          |
|         3 | Sécurité            |
|         4 | Base de données     |
|         4 | Langages            |
|         4 | Système             |
+-----------+---------------------+
15 rows in set (0.00 sec)
```

Les valeurs NULL apparaissent en premier dans un tri croissant.

### d. Limiter le nombre de lignes : clause LIMIT

Une clause LIMIT peut être ajoutée à la requête SELECT pour limiter le nombre de lignes retournées.

*Syntaxe*

```
SELECT [DISTINCT] expression[,...] | *
FROM nom_table
[WHERE conditions]
[ORDER BY expression [ASC | DESC][,...]]LIMIT [offset,] nombre_lignes
```

nombre_lignes est une constante entière qui spécifie le nombre de lignes à retourner.

offset, s'il est présent, est aussi une constante entière qui spécifie le numéro de la première ligne à retourner (0 pour la première ligne). LIMIT n est équivalent à LIMIT 0,n.

Si la requête comporte une clause WHERE et/ou une clause ORDER BY, la clause LIMIT doit figurer après.

La clause LIMIT est évaluée après les clauses WHERE et ORDER BY. Combinée à une clause ORDER BY, la clause LIMIT est très pratique pour les interrogations de type "palmarès" ("Top N").

*Exemple*

```
mysql> -- afficher les trois plus "gros" livres (en nombre de pages)
mysql> SELECT titre,nombre_pages
    -> FROM livre
    -> ORDER BY nombre_pages DESC
    -> LIMIT 3;
+------------------+--------------+
| titre            | nombre_pages |
+------------------+--------------+
| MySQL 5 et PHP 5 |          972 |
| PHP 5.2          |          518 |
| Oracle 10g       |          489 |
+------------------+--------------+
3 rows in set (0.00 sec)

-- afficher les deux suivants
mysql> SELECT titre,nombre_pages
    -> FROM livre
    -> ORDER BY nombre_pages DESC
    -> LIMIT 3,2;
+-------------------+--------------+
| titre             | nombre_pages |
+-------------------+--------------+
| BusinessObjects 6 |          470 |
| MySQL 5           |          468 |
+-------------------+--------------+
2 rows in set (0.00 sec)
```

### e. Lire dans plusieurs tables : jointure

Dans une interrogation, il est souvent nécessaire d'extraire des informations dans plusieurs tables.

Pour cela, il faut lister les tables souhaitées dans la clause FROM et effectuer une jointure entre les tables. Une jointure consiste à indiquer au serveur MySQL comment relier les tables entre elles.

Lister les tables dans la clause FROM et oublier d'écrire la jointure n'est pas une erreur de syntaxe. Le serveur retourne un résultat en effectuant un produit cartésien entre les tables : chaque ligne de la première table est combinée avec chaque ligne de la deuxième table. Un tel résultat est rarement souhaité !

MySQL supporte plusieurs syntaxes pour écrire une jointure. Nous présenterons ici les plus utilisées.

#### Jointure interne

Pour écrire une **jointure interne** ("simple") entre deux tables, vous pouvez utiliser une des trois syntaxes suivantes :

```
FROM nom_table1,nom_table2
WHERE nom_table1.nom_colonne1 = nom_table2.nom_colonne2
```

```
FROM nom_table1 [INNER] JOIN nom_table2 ON nom_table1.nom_colonne1 =
 nom_table2.nom_colonne2
```

```
FROM nom_table1 [INNER] JOIN nom_table2 USING (nom_colonne[,...])
```

Une jointure se matérialise le plus souvent par une condition d'égalité entre une colonne de la première table et une colonne de la deuxième table : c'est une "équi-jointure", jointure la plus souvent utilisée (mais il existe d'autre types de jointures, non basées sur une égalité).

Dans la première syntaxe, la condition de jointure est écrite dans la clause WHERE et les tables sont séparées par une virgule dans la clause FROM. Cette syntaxe, bien qu'elle soit supportée par de nombreux SGBDR, n'est pas la syntaxe de la norme ANSI.

Les deux dernières syntaxes sont conformes à la norme ANSI. Les tables sont jointes dans la clause FROM par l'intermédiaire du mot clé JOIN (INNER est optionnel, c'est la valeur par défaut) et la condition de jointure est, elle aussi, spécifiée dans la clause FROM soit avec une clause ON soit avec une clause USING. Avec la clause ON, la condition de jointure est écrite explicitement (comme dans la clause WHERE de la première syntaxe). Avec la clause USING, une équi-jointure est implicitement réalisée sur les colonnes mentionnées qui doivent donc porter le même nom dans les

deux tables ; `USING (nom_colonne)` est équivalent à `ON nom_table1.nom_colonne = nom_table2.nom_colonne`.

Lorsque vous écrivez des interrogations portant sur plusieurs tables, il est vivement conseillé de définir des alias de tables et de les utiliser pour préfixer le nom des colonnes.

*Exemple avec deux tables*

```
mysql> SELECT col.nom,liv.titre
    -> FROM collection col,livre liv
    -> WHERE col.id = liv.id_collection
    ->       AND liv.annee_parution=2006;
+-------------------------+------------------+
| nom                     | titre            |
+-------------------------+------------------+
| Ressources Informatiques | MySQL 5          |
| TechNote                | PHP 5            |
| Coffret Technique       | MySQL 5 et PHP 5 |
+-------------------------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT col.nom,liv.titre
    -> FROM collection col JOIN livre liv
    ->      ON (col.id = liv.id_collection)
    -> WHERE liv.annee_parution=2006;
+-------------------------+------------------+
| nom                     | titre            |
+-------------------------+------------------+
| Ressources Informatiques | MySQL 5          |
| TechNote                | PHP 5            |
| Coffret Technique       | MySQL 5 et PHP 5 |
+-------------------------+------------------+
3 rows in set (0.00 sec)
```

*Exemple avec trois tables*

```
mysql> SELECT
    ->    liv.id,
    ->    liv.titre,
    ->    aut.nom
    -> FROM
    ->    livre liv,
    ->    auteur_livre aul,
    ->    auteur aut
    -> WHERE
    ->    liv.id = aul.id_livre
    ->    AND aul.id_auteur = aut.id
    ->    AND liv.annee_parution=2006
    -> ORDER BY
    ->    liv.titre;
+----+------------------+---------+
| id | titre            | nom     |
+----+------------------+---------+
|  7 | MySQL 5          | THIBAUD |
|  9 | MySQL 5 et PHP 5 | HEURTEL |
|  9 | MySQL 5 et PHP 5 | THIBAUD |
|  2 | PHP 5            | HEURTEL |
+----+------------------+---------+
4 rows in set (0.00 sec)

mysql> SELECT
    ->    liv.id,
    ->    liv.titre,
    ->    aut.nom
    -> FROM
    ->    livre liv
    ->       JOIN
    ->    auteur_livre aul
```

```
    ->       ON (liv.id = aul.id_livre)
    ->       JOIN
    ->    auteur aut
    ->       ON (aul.id_auteur = aut.id)
    -> WHERE
    ->    liv.annee_parution=2006
    -> ORDER BY
    ->    liv.titre;
+----+-----------------+---------+
| id | titre           | nom     |
+----+-----------------+---------+
|  7 | MySQL 5         | THIBAUD |
|  9 | MySQL 5 et PHP 5 | HEURTEL |
|  9 | MySQL 5 et PHP 5 | THIBAUD |
|  2 | PHP 5           | HEURTEL |
+----+-----------------+---------+
4 rows in set (0.00 sec)
```

## Jointure externe

Dans une **jointure interne**, seules les lignes en correspondance dans les deux tables sont retournées : si une ligne de la première table n'a pas de correspondance dans la deuxième table, elle n'est pas retournée (et réciproquement).

*Exemple*

```
mysql> SELECT titre,id_promotion FROM livre
    -> WHERE annee_parution = 2006;
+------------------+--------------+
| titre            | id_promotion |
+------------------+--------------+
| PHP 5            |            1 |
| MySQL 5          |         NULL |
| MySQL 5 et PHP 5 |            2 |
+------------------+--------------+
3 rows in set (0.00 sec)

mysql> SELECT liv.titre,pro.intitule
    -> FROM livre liv JOIN promotion pro
    ->      ON (liv.id_promotion = pro.id)
    -> WHERE liv.annee_parution = 2006;
+------------------+-------------------------------------+
| titre            | intitule                            |
+------------------+-------------------------------------+
| PHP 5            | -5% sur cet ouvrage                 |
| MySQL 5 et PHP 5 | Frais de port offerts sur cet ouvrage |
+------------------+-------------------------------------+
2 rows in set (0.00 sec)
```

Sur cet exemple, le livre "MySQL 5" n'est pas retourné par la deuxième requête car il n'a pas de promotion.

Pour retourner toutes les lignes d'une des deux tables même si elles n'ont pas de correspondance, il faut écrire une **jointure externe**. Avec une jointure externe, le serveur MySQL crée une ligne avec des valeurs NULL pour la table qui n'a pas de ligne en correspondance avec l'autre table ; cette ligne "vide" est utilisée pour établir la "correspondance" avec l'autre table.

---

Lorsque vous écrivez une jointure, posez-vous toujours la question de savoir s'il peut ne pas y avoir de correspondance entre les deux tables, et si c'est le cas, ce que vous souhaitez faire : conserver ou non les lignes en question.

---

Pour écrire une **jointure externe** entre deux tables, vous pouvez utiliser une des quatre syntaxes suivantes :

```
FROM nom_table1 LEFT [OUTER]  JOIN nom_table2
ON nom_table1.nom_colonne1 = nom_table2.nom_colonne2

FROM nom_table1 LEFT [OUTER] JOIN nom_table2
USING (nom_colonne[,...])
```

```
FROM nom_table1 RIGHT [OUTER]  JOIN nom_table2
ON nom_table1.nom_colonne1 = nom_table2.nom_colonne2

FROM nom_table1 RIGHT [OUTER] JOIN nom_table2
USING (nom_colonne[,...])
```

Le mot clé OUTER est optionnel car le sens de la jointure externe doit être précisé avec le mot clé LEFT ou RIGHT. Dans une jointure "gauche" nom_table1 LEFT JOIN nom_table2, les lignes de la table de "gauche" nom_table1 sont retournées même si elles n'ont pas de correspondance dans l'autre table. Dans une jointure "droite" nom_table1 RIGHT JOIN nom_table2, les lignes de la table de "droite" nom_table2 sont retournées même si elles n'ont pas de correspondance dans l'autre table.

Comme pour la jointure interne, la condition de jointure peut être spécifiée à l'aide d'une clause ON ou USING.

*Exemple*

```
mysql> SELECT liv.titre,pro.intitule
    -> FROM livre liv LEFT JOIN promotion pro
    ->      ON (liv.id_promotion = pro.id)
    -> WHERE liv.annee_parution = 2006;
+------------------+--------------------------------------+
| titre            | intitule                             |
+------------------+--------------------------------------+
| PHP 5            | -5% sur cet ouvrage                  |
| MySQL 5          | NULL                                 |
| MySQL 5 et PHP 5 | Frais de port offerts sur cet ouvrage |
+------------------+--------------------------------------+
3 rows in set (0.00 sec)
```

Maintenant, le livre "MySQL 5" est bien retourné dans le résultat, bien qu'il n'ait pas de promotion.

Si la requête comporte une condition supplémentaire sur la table externe (celle sur laquelle une ligne vide est générée pour la mise en correspondance), il faut la mettre dans la clause ON de la jointure et non dans la clause WHERE, sous peine d'obtenir une jointure simple à la place de la jointure externe.

*Exemple*

```
mysql> SELECT liv.titre,pro.intitule
    -> FROM livre liv LEFT JOIN promotion pro
    ->      ON (liv.id_promotion = pro.id)
    -> WHERE liv.annee_parution = 2006
    ->       AND pro.est_active = TRUE;
+-------+---------------------+
| titre | intitule            |
+-------+---------------------+
| PHP 5 | -5% sur cet ouvrage |
+-------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT liv.titre,pro.intitule
    -> FROM livre liv LEFT JOIN promotion pro
    ->      ON (liv.id_promotion = pro.id
    ->          AND pro.est_active = TRUE)
    -> WHERE liv.annee_parution = 2006;
+------------------+---------------------+
| titre            | intitule            |
+------------------+---------------------+
| PHP 5            | -5% sur cet ouvrage |
| MySQL 5          | NULL                |
| MySQL 5 et PHP 5 | NULL                |
+------------------+---------------------+
3 rows in set (0.00 sec)
```

## Auto jointure

Il est possible de joindre une table avec elle-même (notion d'**auto jointure**). Pour cela, il faut faire figurer la table deux fois dans la clause FROM, avec des alias de table différents, et écrire la jointure entre les deux tables comme s'il s'agissait de tables différentes. Cette technique est particulièrement utile pour interroger une table qui implémente une structure hiérarchique.

*Exemple*

```
mysql> SELECT par.titre rubrique,enf.titre sous_rubrique
    -> FROM rubrique par,rubrique enf
    -> WHERE enf.id_parent = par.id;
+-----------------+-----------------+
| rubrique        | sous_rubrique   |
+-----------------+-----------------+
| Base de données | MySQL           |
| Base de données | Oracle          |
| Développement   | Langages        |
| Développement   | Méthode         |
| Internet        | HTML - XML      |
| Internet        | Conception Web  |
| Internet        | Sécurité        |
| Open Source     | Système         |
| Open Source     | Langages        |
| Open Source     | Base de données |
+-----------------+-----------------+
10 rows in set (0.00 sec)
```

## 5. Ajouter des lignes dans une table

L'ordre SQL INSERT permet d'ajouter des lignes dans une table.

*Syntaxe 1*

```
INSERT [IGNORE]
[INTO] nom_table [(nom_colonne,...)]
VALUES ({expression | DEFAULT},...),(...),...
[ ON DUPLICATE KEY UPDATE nom_colonne=expression, ... ]
```

*Syntaxe 2*

```
INSERT [IGNORE]
[INTO] nom_table
SET nom_colonne ={ expression | DEFAULT}, ...
[ ON DUPLICATE KEY UPDATE nom_colonne= expression, ... ]
```

nom_table est le nom de la table dans laquelle les données sont ajoutées.

Les colonnes concernées par l'insertion sont spécifiées soit par une liste de noms de colonnes dernière le nom de la table (première syntaxe), soit par la clause SET (deuxième syntaxe). Dans la première syntaxe, si la liste des colonnes est absente, toutes les colonnes de la table sont concernées par défaut. Il faut noter qu'il n'y a pas d'obligation à insérer une valeur dans toutes les colonnes de la table.

Les valeurs des colonnes sont spécifiées soit par la clause VALUES (première syntaxe), soit par la clause SET (deuxième syntaxe). Dans la première syntaxe, la clause VALUES doit comporter une expression pour chaque colonne mentionnée dans la liste des colonnes (dans l'ordre) ; si la liste des colonnes est absente, la clause VALUES doit fournir une valeur pour chaque colonne de la table, dans l'ordre des colonnes de la table. Dans la deuxième syntaxe, la valeur de chaque colonne est donnée par l'expression située à droite du signe =. Dans la première syntaxe, plusieurs listes de valeurs peuvent être spécifiées (séparées par des virgules) afin d'insérer plusieurs lignes en une seule opération.

Dans la première syntaxe, pour du code en production, il est conseillé de toujours spécifier la liste des colonnes, afin que la requête puisse continuer à fonctionner même si des colonnes sont ajoutées dans la table.

*Exemples*

```
mysql> DESC collection;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom      | varchar(25)  | NO   | UNI |         |                |
| prix_ht  | decimal(5,2) | YES  |     | 20.00   |                |
| frais_ht | decimal(5,2) | YES  |     | NULL    |                |
```

```
+----------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> INSERT INTO collection
    -> VALUES(5,'Solutions Informatiques',36.97,1.25);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO collection(nom)
    -> VALUES('ExpertIT');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM collection WHERE id >= 5;
+----+-------------------------+---------+----------+
| id | nom                     | prix_ht | frais_ht |
+----+-------------------------+---------+----------+
|  5 | Solutions Informatiques |   36.97 |     1.25 |
|  6 | ExpertIT                |   20.00 |     NULL |
+----+-------------------------+---------+----------+
2 rows in set (0.00 sec)
```

Comme le montre cet exemple, si une colonne n'est pas mentionnée dans l'insertion, mais qu'elle a été définie avec l'attribut AUTO_INCREMENT (cf. chapitre Construire une base de données dans MySQL - Gérer les tables), un entier unique lui est affecté. Si ce n'est pas le cas; la valeur par défaut de la colonne lui est affectée ; le mot clé DEFAULT permet d'affecter explicitement la valeur par défaut de la colonne.

Comme nous le verrons dans le chapitre Construire une base de données dans MySQL, la valeur par défaut d'une colonne peut être définie lors de la création de la table. Lors de l'insertion, si aucune valeur par défaut n'est définie explicitement pour une colonne et que la colonne autorise les valeurs NULL, la valeur NULL est affectée comme valeur par défaut. Par contre, si la colonne est obligatoire, le fonctionnement dépend du mode SQL actif :

- Si le mode SQL strict n'est pas actif (cas par défaut), MySQL utilise sa propre valeur par défaut liée au type de données : 0 pour un nombre, chaîne vide pour une chaîne, date "zéro" pour une date.

- Si le mode SQL strict est actif, une erreur se produit.

*Exemples*

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> INSERT INTO collection()  -- un peu bizarre comme INSERT
    -> VALUES();                 -- mais c'est pour tester !
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+------------------------------------------+
| Level   | Code | Message                                  |
+---------+------+------------------------------------------+
| Warning | 1364 | Field 'nom' doesn't have a default value |
+---------+------+------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT nom,prix_ht FROM collection
    -> WHERE id = last_insert_id();
+-----+---------+
| nom | prix_ht |
+-----+---------+
|     |   20.00 |
+-----+---------+
1 row in set (0.00 sec)

mysql> SET @@sql_mode=STRICT_ALL_TABLES;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO collection()
    -> VALUES();
ERROR 1364 (HY000): Field 'nom' doesn't have a default value

mysql> SET @@sql_mode='';
Query OK, 0 rows affected (0.00 sec)
```

---

La fonction `last_insert_id` utilisée dans cet exemple retourne la valeur automatiquement générée par une colonne de type `AUTO_INCREMENT` lors du dernier `INSERT` (cf. chapitre Utiliser les fonctions MySQL - Fonctions système).

---

Le comportement de MySQL vis-à-vis de l'insertion d'une valeur invalide (date inexistante par exemple) ou trop grande (`Olivier` dans un `VARCHAR(4)`) dépend aussi du mode SQL :

- 
    - une chaîne trop grande est tronquée ;
    - une date invalide est remplacée par une date "zéro" ;
    - un nombre trop grand est remplacé par la valeur maximum autorisée par le type.
- Si le mode SQL strict est actif, une erreur est générée.

*Exemples*

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> INSERT INTO collection(nom,prix_ht)
    -> VALUES('Coffret Technique Certification',1234);
Query OK, 1 row affected, 2 warnings (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+---------------------------------------------------------+
| Level   | Code | Message                                                 |
+---------+------+---------------------------------------------------------+
| Warning | 1265 | Data truncated for column 'nom' at row 1                |
| Warning | 1264 | Out of range value adjusted for column 'prix_ht' at row 1 |
+---------+------+---------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT nom,prix_ht FROM collection
    -> WHERE id = last_insert_id();
+-------------------------+---------+
| nom                     | prix_ht |
+-------------------------+---------+
| Coffret Technique Certifi |  999.99 |
+-------------------------+---------+
1 row in set (0.00 sec)

mysql> SET @@sql_mode=<$I[]STRICT_ALL_TABLES> STRICT_ALL_TABLES;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO collection(nom,prix_ht)
    -> VALUES('Coffret Technique Certification',1234);
ERROR 1406 (22001): Data too long for column 'nom' at row 1

mysql> SET @@sql_mode='';
Query OK, 0 rows affected (0.00 sec)
```

## Clause IGNORE

La clause `IGNORE` permet d'ignorer les erreurs liées à l'insertion d'une ligne qui provoquerait un doublon dans une clé primaire ou unique ; si le cas se produit, la ligne n'est pas insérée et aucun message d'erreur n'est affiché.

*Exemples*

```
mysql> INSERT INTO collection(id,nom,prix_ht)
    -> VALUES(1,'Epsilon',51.18);
ERROR 1062 (23000): Duplicate entry '1' for key 1

mysql>  INSERT IGNORE INTO collection(id,nom,prix_ht)
    -> VALUES(1,'Epsilon',51.18);
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT nom FROM collection WHERE id = 1;
+-------------------------+
| nom                     |
+-------------------------+
| Ressources Informatiques |
+-------------------------+
1 row in set (0.00 sec)
```

## Clause ON DUPLICATE KEY UPDATE

La clause `ON DUPLICATE KEY UPDATE` permet de transformer en `UPDATE` l'insertion d'une ligne qui provoque un doublon sur une clé primaire ou unique.

Dans la clause `UPDATE`, l'expression `VALUES(nom_colonne)` permet de faire référence à la valeur de la colonne dans la clause `INSERT` d'origine.

*Exemples*

```
mysql> INSERT INTO auteur(nom,prenom,mail)
    -> VALUES('HEURTEL','Olivier','contact@olivier-heurtel.fr');
ERROR 1062 (23000): Duplicate entry 'HEURTEL-Olivier' for key 2

mysql>  INSERT INTO auteur(nom,prenom,mail)
    -> VALUES('HEURTEL','Olivier','contact@olivier-heurtel.fr')
    -> ON DUPLICATE KEY UPDATE mail = VALUES(mail);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT mail FROM auteur WHERE nom = 'HEURTEL';
+----------------------------+
| mail                       |
+----------------------------+
| contact@olivier-heurtel.fr |
+----------------------------+
1 row in set (0.00 sec)
```

---

Attention ! Si la ligne insérée provoque un doublon sur deux lignes (par exemple sur la clé primaire pour la première ligne et sur une clé unique pour la deuxième ligne), seule une ligne sera mise à jour. Il est conseillé d'éviter ce genre de situation.

---

## Insertion de plusieurs lignes

Dans la première syntaxe de l'ordre `INSERT`, plusieurs listes de valeurs peuvent être spécifiées (séparées par des virgules) afin d'insérer plusieurs lignes en une seule opération.

Si une erreur se produit lors d'un `INSERT` multilignes, l'ordre est interrompu et les lignes déjà insérées sont conservées ou pas selon le type de table :

- pour une table non transactionnelle (c'est le cas par défaut en général), les lignes déjà insérées sont conservées ;
- pour une table transactionnelle (cf. chapitre Techniques avancées avec MySQL - Gérer les transactions et les

accès concurrents), les lignes déjà insérées sont annulées.

*Exemple avec une table non transactionnelle*

```
mysql> INSERT INTO
    ->    auteur(nom,prenom)
    -> VALUES
    ->    ('NOIRAULT','Claire'),
    ->    ('GABILLAUD','Jérôme'),
    ->    ('HEURTEL','Olivier'), -- existe déjà !!
    ->    ('HUGO','Victor');
ERROR 1062 (23000): Duplicate entry 'HEURTEL-Olivier' for key 2

mysql> SELECT id,nom,prenom FROM auteur WHERE id > 3;
+----+-----------+--------+
| id | nom       | prenom |
+----+-----------+--------+
|  4 | NOIRAULT  | Claire |
|  5 | GABILLAUD | Jérôme |
+----+-----------+--------+
2 rows in set (0.00 sec)
```

L'auteur "Victor Hugo" n'a pas été inséré dans la table.

Dans le chapitre Techniques avancées avec MySQL - Utiliser des sous-requêtes, nous verrons comment ajouter des lignes dans une table à l'aide d'une autre requête (sous-requête).

## 6. Modifier des lignes dans une table

L'ordre SQL UPDATE permet de modifier des lignes dans une table.

*Syntaxe*

```
UPDATE [IGNORE] nom_table
SET nom_colonne={expression | DEFAULT} [,...]
[WHERE condition]
[ORDER BY tri]
[LIMIT nombre_lignes]
```

nom_table est le nom de la table dans laquelle les données sont modifiées.

La clause SET indique les colonnes à modifier avec leur nouvelle valeur. Il faut noter qu'il n'y a pas d'obligation à modifier toutes les colonnes de la table.

Le mot clé DEFAULT permet d'affecter explicitement la valeur par défaut de la colonne. Comme dans le cas de l'insertion, si la valeur DEFAULT est mentionnée, mais qu'aucune valeur par défaut n'est définie explicitement sur la colonne et que la colonne est obligatoire, MySQL utilise sa propre valeur par défaut liée au type de données (0 pour un nombre, chaîne vide pour une chaîne, date "zéro" pour une date).

La clause WHERE est optionnelle. Si elle est omise, toutes les lignes de la table sont modifiées. Si elle est spécifiée, seules les lignes vérifiant la condition sont mises à jour. La syntaxe pour écrire une condition est la même que pour la clause WHERE de l'ordre SELECT.

*Exemples*

```
mysql> UPDATE auteur SET mail = 'olivier.heurtel@gmail.com'
    -> WHERE nom = 'HEURTEL' AND prenom = 'Olivier';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE collection SET nom = 'Epsilon',prix_ht=51.18
    -> WHERE nom = '';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE collection
```

```
    -> SET prix_ht = ROUND(prix_ht*1.015,2) -- augmentation de 1,5%
    -> WHERE nom = 'Epsilon';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE livre SET niveau='Confirmé' WHERE id_collection = 2;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Si la valeur d'une colonne est inchangée, MySQL ne fait pas la mise à jour ; il peut donc y avoir une différence entre le nombre de lignes sélectionnées et le nombre de lignes modifiées.

*Exemple*

```
mysql> UPDATE livre SET id_promotion=1 WHERE id_collection=1;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 5  Changed: 3  Warnings: 0
```

Sur cet exemple, deux livres de la collection n° 1 ont déjà le code promotion n° 1 ; en conséquence, la requête ne met réellement à jour que trois livres parmi les cinq de la collection.

Le comportement vis-à-vis des valeurs invalides ou trop grandes, ou vis-à-vis des erreurs dans un UPDATE multilignes est le même que pour l'ordre INSERT.

*Exemple*

```
mysql> UPDATE collection SET prix_ht = 5200 WHERE nom = 'Epsilon';
Query OK, 1 row affected, 1 warning (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 1

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------------------------+
| Level   | Code | Message                                                  |
+---------+------+----------------------------------------------------------+
| Warning | 1264 | Out of range value adjusted for column 'prix_ht' at row 1 |
+---------+------+----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT prix_ht FROM collection WHERE nom = 'Epsilon';
+---------+
| prix_ht |
+---------+
|  999.99 |
+---------+
1 row in set (0.00 sec)
```

## Clauses ORDER BY et LIMIT

Si une clause ORDER BY est spécifiée, les lignes sont modifiées dans l'ordre défini par la clause. Dans la pratique, cette clause ne présente un intérêt que lorsqu'elle est utilisée avec la clause LIMIT.

La clause LIMIT permet de limiter le nombre de lignes sélectionnées (avant la version 4.0.13, cette clause limitait le nombre de lignes modifiées).

*Exemple : augmenter le prix de la collection la moins chère*

```
mysql> SELECT nom,prix_ht FROM collection ORDER BY prix_ht LIMIT 1;
+----------+---------+
| nom      | prix_ht |
+----------+---------+
| TechNote |    9.48 |
+----------+---------+
1 row in set (0.00 sec)

mysql>  UPDATE collection SET prix_ht = prix_ht + 0.50
    -> ORDER BY prix_ht LIMIT 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT nom,prix_ht FROM collection ORDER BY prix_ht LIMIT 1;
```

```
+----------+---------+
| nom      | prix_ht |
+----------+---------+
| TechNote |    9.98 |
+----------+---------+
1 row in set (0.01 sec)
```

### Clause IGNORE

La clause `IGNORE` permet d'ignorer les erreurs liées à la modification de lignes qui provoquerait un doublon dans une clé primaire ou unique ; si le cas se produit, la ligne n'est pas modifiée, aucun message d'erreur n'est affiché et la modification se poursuit sur les autres lignes.

*Exemple*

```
mysql> UPDATE collection SET nom = 'TechNote' WHERE id = 1;
ERROR 1062 (23000): Duplicate entry 'TechNote' for key 2

mysql>  UPDATE IGNORE collection SET nom = 'TechNote' WHERE id = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT nom FROM collection WHERE id = 1;
+-------------------------+
| nom                     |
+-------------------------+
| Ressources Informatiques |
+-------------------------+
1 row in set (0.00 sec)
```

### Mise à jour dans plusieurs tables jointes

MySQL supporte aussi une syntaxe de mise à jour dans plusieurs tables jointes.

*Exemple*

```
mysql> SELECT col.nom,col.prix_ht,liv.isbn,liv.id_promotion
    -> FROM livre liv JOIN collection col
    ->      ON (liv.id_collection = col.id)
    -> WHERE
    ->    col.nom = 'TechNote';
+----------+---------+---------------+--------------+
| nom      | prix_ht | isbn          | id_promotion |
+----------+---------+---------------+--------------+
| TechNote |    9.98 | 2-7460-3104-3 |            1 |
| TechNote |    9.98 | 2-7460-2834-4 |         NULL |
| TechNote |    9.98 | 2-7460-2833-6 |         NULL |
+----------+---------+---------------+--------------+
2 rows in set (0.00 sec)

mysql> UPDATE
    ->    livre liv JOIN collection col
    ->    ON (liv.id_collection = col.id)
    -> SET
    ->    -- augmenter le prix de la collection
    ->    col.prix_ht = col.prix_ht + 0.5,
    ->    -- mettre les livres (de la collection) en promotion
    ->    liv.id_promotion = 2
    -> WHERE
    ->    col.nom = 'TechNote'
    ->    AND liv.id_promotion IS NULL; -- pas déjà en promotion
Query OK, 3 rows affected (0.01 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT col.nom,col.prix_ht,liv.isbn,liv.id_promotion
    -> FROM livre liv JOIN collection col
    ->      ON (liv.id_collection = col.id)
    -> WHERE
```

```
    ->    col.nom = 'TechNote';
+----------+---------+---------------+--------------+
| nom      | prix_ht | isbn          | id_promotion |
+----------+---------+---------------+--------------+
| TechNote |   10.48 | 2-7460-3104-3 |            1 |
| TechNote |   10.48 | 2-7460-2834-4 |            2 |
| TechNote |   10.48 | 2-7460-2833-6 |            2 |
+----------+---------+---------------+--------------+
3 rows in set (0.00 sec)
```

Sur cet exemple, les tables collection et livre sont mises à jour avec un seul ordre UPDATE : le prix de la collection "TechNote" est augmenté et les livres de cette collection qui n'étaient pas déjà en promotion sont mis en promotion. Il y a bien trois lignes mises à jour : une dans la table collection et deux dans la table livre.

Les différentes syntaxes de jointure (interne et externe) présentées précédemment sont supportées. Pour plus d'informations sur cette syntaxe, consultez la documentation.

## 7. Supprimer des lignes dans une table

L'ordre SQL DELETE permet de supprimer des lignes dans une table.

*Syntaxe*

```
DELETE [IGNORE] FROM nom_table
[WHERE condition]
[ORDER BY tri]
[LIMIT nombre_lignes]
```

nom_table est le nom de la table dans laquelle les données sont supprimées.

La clause WHERE est optionnelle. Si elle est omise, toutes les lignes de la table sont supprimées. Si elle est spécifiée, seules les lignes vérifiant la condition sont supprimées. La syntaxe pour écrire une condition est la même que pour la clause WHERE de l'ordre SELECT.

*Exemples*

```
mysql> DELETE FROM promotion WHERE id = 3;
Query OK, 1 row affected (0.00 sec)

mysql> DELETE FROM collection WHERE prix_ht = 999.99;
Query OK, 2 rows affected (0.00 sec)
```

### Clauses ORDER BY et LIMIT

Si une clause ORDER BY est spécifiée, les lignes sont supprimées dans l'ordre défini par la clause. Dans la pratique, cette clause ne présente un intérêt que lorsqu'elle est utilisée avec la clause LIMIT.

La clause LIMIT permet de limiter le nombre de lignes supprimées.

*Exemple : supprimer la "dernière" ligne (plus grand identifiant)*

```
mysql> SELECT id,titre FROM rubrique ORDER BY id DESC LIMIT 1;
+----+----------------------+
| id | titre                |
+----+----------------------+
| 15 | Messagerie et travai |
+----+----------------------+
1 row in set (0.00 sec)

mysql>  DELETE FROM rubrique ORDER BY id DESC LIMIT 1;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT id,titre FROM rubrique ORDER BY id DESC LIMIT 1;
+----+-----------------+
| id | titre           |
+----+-----------------+
| 14 | Base de données |
+----+-----------------+
```

```
1 row in set (0.00 sec)
```

### Clause IGNORE

La clause `IGNORE` permet d'ignorer les erreurs liées à la suppression de lignes qui provoqueraient une erreur (violation de clé étrangère - cf. chapitre Construire une base de données dans MySQL - Utiliser les clés et les index) ; si le cas se produit, la ligne n'est pas supprimée, aucun message d'erreur n'est affiché et la suppression se poursuit sur les autres lignes.

### Suppression dans plusieurs tables jointes

MySQL supporte aussi une syntaxe de suppression dans plusieurs tables jointes.

*Exemple*

```
mysql> SELECT id,isbn FROM livre WHERE id = 1;
+----+---------------+
| id | isbn          |
+----+---------------+
|  1 | 2-7460-1451-3 |
+----+---------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM rubrique_livre WHERE id_livre = 1;
+-------------+----------+
| id_rubrique | id_livre |
+-------------+----------+
|           7 |        1 |
|          10 |        1 |
|          13 |        1 |
+-------------+----------+
3 rows in set (0.01 sec)

mysql> SELECT * FROM auteur_livre WHERE id_livre = 1;
+-----------+----------+
| id_auteur | id_livre |
+-----------+----------+
|         1 |        1 |
+-----------+----------+
1 row in set (0.00 sec)

mysql> DELETE FROM
    ->    liv,rul,aul -- alias définis dans la clause USING
    -> USING
    ->    livre liv,rubrique_livre rul,auteur_livre aul
    -> WHERE
    ->    liv.id = rul.id_livre
    ->    AND liv.id = aul.id_livre
    ->    AND liv.isbn = '2-7460-1451-3';
Query OK, 5 rows affected (0.00 sec)

mysql> SELECT id,isbn FROM livre WHERE id = 1;
Empty set (0.01 sec)

mysql> SELECT * FROM rubrique_livre WHERE id_livre = 1;
Empty set (0.01 sec)

mysql> SELECT * FROM auteur_livre WHERE id_livre = 1;
Empty set (0.00 sec)
```

Sur cet exemple, l'ordre `DELETE` supprime une ligne dans la table `livre`, ainsi que les lignes jointes dans les tables `rubrique_livre` et `auteur_livre` (ces lignes n'ont plus lieu d'être).

La clause `FROM` spécifie les tables dans lesquelles des lignes doivent être supprimées. La clause `USING` spécifie toutes les tables utilisées dans la jointure ; il peut y avoir dans la clause `USING` des tables non présentes dans la clause `FROM` (tables utilisées en jointure mais dans lesquelles il n'y a pas de lignes à supprimer).

Si un alias de table est défini dans la clause `USING` alors cet alias doit être utilisé dans la clause `FROM`.

Les différentes syntaxes de jointure (interne et externe) présentées précédemment sont supportées. Une autre

syntaxe de suppression multi-table peut être utilisée. Pour plus d'informations sur les différentes syntaxes de suppression multi-table, consultez la documentation.

# 8. Exporter et importer des données

## a. Exporter des données

L'ordre SQL SELECT peut contenir une clause INTO OUTFILE qui permet de stocker le résultat de la requête dans un fichier.

*Syntaxe*

```
INTO OUTFILE 'nom_fichier'
[FIELDS [TERMINATED BY 'caractères']
        [[OPTIONALLY] ENCLOSED BY 'caractère']
        [ESCAPED BY 'caractère' ]]
[LINES [STARTING BY 'caractères'] [TERMINATED BY 'caractères']]
```

La clause INTO OUTFILE, si elle est présente, doit être définie juste avant la clause FROM.

Pour utiliser cette clause, l'utilisateur doit avoir le privilège FILE (cf. chapitre 4 - B - Gérer les utilisateurs et les droits).

Le fichier dans lequel les données sont écrites ne doit pas déjà exister. De plus, le serveur MySQL doit avoir les droits nécessaires pour écrire dans le fichier. Sur une plate-forme Windows, pour spécifier le chemin du fichier, vous devez au choix échapper l'anti-slash (`c:\\temp\\data.txt`) ou utiliser le slash (`c:/temp/data.txt`).

La clause optionnelle FIELDS permet de définir les options suivantes :

TERMINATED BY

Délimiteur de fin de champ (plusieurs caractères autorisés, tabulation par défaut : \t).

[OPTIONALLY] ENCLOSED BY

Caractère utilisé pour entourer les champs (un seul caractère autorisé, vide par défaut). Si le mot clé OPTIONALLY est absent, seules les champs de type texte sont entourés ; s'il est présent, tous les champs sont entourés.

ESCAPED BY

Caractère utilisé pour échapper les caractères suivants (un seul caractère autorisé, anti-slash par défaut : \\) :

- le caractère lui-même ;
- le caractère défini par l'option ENCLOSED BY ;
- le premier caractère des options TERMINATED BY (FIELDS et LINES) ;
- le caractère de code ASCII 0.

Si la clause FIELDS est omise, les valeurs par défaut sont les suivantes :

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

La clause optionnelle LINES permet de définir les options suivantes :

STARTING BY

Délimiteur de début de ligne (plusieurs caractères autorisés, vide par défaut).

TERMINATED BY

Délimiteur de fin de ligne (plusieurs caractères autorisés, retour à la ligne par défaut : \n).

Si la clause LINES est omise, les valeurs par défaut sont les suivantes :

```
LINES TERMINATED BY '\n'
```

La manière dont les valeurs NULL sont écrites dans le fichier dépend de la valeur de l'option ESCAPED BY :

- Si l'option est vide, le mot NULL est écrit dans le champ.
- Si l'option n'est pas vide, la lettre N précédée du caractère d'échappement défini par l'option est écrite dans le champ (\N par exemple).

*Exemple*

```
SELECT liv.isbn,liv.annee_parution,pro.intitule
FROM livre liv LEFT JOIN promotion pro
     ON liv.id_promotion = pro.id
INTO OUTFILE '/tmp/liste-livre-1.txt';
```

*Résultat dans le fichier liste-livre-1.txt*

```
978-2-7460-3992-6       2007    -5% sur cet ouvrage
2-7460-3104-3   2006    -5% sur cet ouvrage
2-7460-2778-X   2005    -5% sur cet ouvrage
2-7460-2834-4   2005    Frais de port offerts sur cet ouvrage
2-7460-2833-6   2005    Frais de port offerts sur cet ouvrage
2-7460-2281-8   2004    -5% sur cet ouvrage
2-7460-3004-7   2006    -5% sur cet ouvrage
2-7460-2340-7   2004    \N
2-7460-3377-1   2006    Frais de port offerts sur cet ouvrage
```

*Exemple*

```
SELECT liv.isbn,liv.annee_parution,pro.intitule
FROM livre liv LEFT JOIN promotion pro
     ON liv.id_promotion = pro.id
INTO OUTFILE '/tmp/liste-livre-2.txt '
FIELDS TERMINATED BY '|'
       OPTIONALLY ENCLOSED BY '"'
LINES STARTING BY '->';
```

*Résultat dans le fichier liste-livre-2.txt*

```
->"978-2-7460-3992-6"|2007|"-5% sur cet ouvrage"
->"2-7460-3104-3"|2006|"-5% sur cet ouvrage"
->"2-7460-2778-X"|2005|"-5% sur cet ouvrage"
->"2-7460-2834-4"|2005|"Frais de port offerts sur cet ouvrage"
->"2-7460-2833-6"|2005|"Frais de port offerts sur cet ouvrage"
->"2-7460-2281-8"|2004|"-5% sur cet ouvrage"
->"2-7460-3004-7"|2006|"-5% sur cet ouvrage"
->"2-7460-2340-7"|2004|\N
->"2-7460-3377-1"|2006|"Frais de port offerts sur cet ouvrage"
```

### b. Importer des données

L'ordre SQL LOAD DATA permet d'importer des données dans une table.

*Syntaxe*

```
LOAD DATA [LOCAL] INFILE 'nom_fichier'
   [REPLACE | IGNORE]
   INTO TABLE nom_table
   [FIELDS [TERMINATED BY 'caractères']
           [[OPTIONALLY] ENCLOSED BY 'caractère']
           [ESCAPED BY 'caractère']]
   [LINES [STARTING BY 'caractères'] [TERMINATED BY 'caractères']]
   [IGNORE nombre LINES]
```

```
[(nom_colonne[,...])]
[SET nom_colonne = expression[,...]]
```

Pour utiliser cette commande, l'utilisateur doit avoir le privilège FILE (cf. chapitre Construire une base de données dans MySQL - Gérer les utilisateurs et les droits).

Si le mot clé LOCAL est spécifié, le fichier est lu à partir de la machine cliente ; sinon, le fichier est lu sur le serveur. Selon le cas, le client MySQL ou le serveur MySQL doit avoir le droit de lire le fichier. Sur une plate-forme Windows, pour spécifier le chemin du fichier, vous devez au choix échapper l'anti-slash (c:\\temp\\data.txt) ou utiliser le slash (c:/temp/data.txt).

La clause IGNORE ou REPLACE indique à MySQL s'il doit ignorer les lignes qui ont la même clé primaire ou unique que des enregistrements déjà présents dans la table, ou s'il doit remplacer les enregistrements en question.

Les clauses FIELDS et LINES sont identiques à celles de la clause INTO OUTFILE présentée précédemment.

La clause IGNORE permet d'ignorer un certain nombre de lignes au début du fichier (ligne de titre par exemple).

La liste de noms de colonnes permet de définir les colonnes dans lesquelles MySQL va charger le contenu des différents champs. Si la liste est omise, les champs seront chargés dans toutes les colonnes de la table, dans l'ordre des colonnes de la table. S'il n'y a pas suffisamment de champs dans la ligne chargée, les colonnes restantes seront alimentées avec la valeur par défaut de la colonne.

La clause SET peut être utilisée pour affecter à des colonnes de la table des valeurs qui sont calculées à l'aide d'une expression. Cette expression peut référencer d'autres colonnes de la table qui sont alimentées directement par le fichier. Dans la liste des colonnes, il est possible de spécifier un nom de variable utilisateur à la place d'un nom de colonne, et d'utiliser cette variable dans la clause SET. Spécifier un nom de variable dans la liste des colonnes et ne pas utiliser cette variable est utile pour ignorer un champ du fichier.

*Exemple*

- Contenu du fichier catalogue.txt :

```
titre|code|prix_ht
Oracle 10g - Administration|RI410GORAA|33.18
PHP 5 - L'accès aux données|TE5PHPAD|9.48
```

- Chargement dans la table catalogue :

```
mysql> LOAD DATA INFILE '/tmp/catalogue.txt' INTO TABLE catalogue
    -> FIELDS TERMINATED BY '|' -- séparateur de champ
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 LINES -- ignorer la ligne de titre
    -> (titre,code,@prix_ht) -- charger le prix HT dans une variable
    -> -- calculer le prix TTC stocké dans la table
    -> -- à partir du prix HT
    -> SET prix_ttc = @prix_ht+ROUND(@prix_ht*5.5/100,2);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM catalogue;
+------------+-----------------------------+----------+
| code       | titre                       | prix_ttc |
+------------+-----------------------------+----------+
| RI410GORAA | Oracle 10g - Administration |    35.00 |
| TE5PHPAD   | PHP 5 - L'accès aux données |    10.00 |
+------------+-----------------------------+----------+
2 rows in set (0.00 sec)
```

# Introduction

Comme nous l'avons vu à plusieurs reprises depuis le début de cet ouvrage, nous pouvons utiliser des expressions dans les différentes clauses des ordres SQL. Une expression peut être écrite en utilisant des colonnes, des expressions littérales, des opérateurs et des fonctions SQL.

Dans ce chapitre, nous allons présenter les fonctions SQL les plus souvent utilisées. Nous ne présenterons pas toutes les fonctions existantes (il y en a plus de 250 !). Nous ne présenterons pas non plus systématiquement toutes les options possibles d'une fonction. Pour en savoir plus, reportez vous à la documentation SQL.

La syntaxe générale d'une fonction est la suivante :

```
nom_fonction([argument][,...])
```

`argument` peut être toute expression dont la valeur est passée en paramètre à la fonction ; `argument` peut lui-même appeler d'autres fonctions.

---

Rappel : sauf indication contraire, une expression qui contient un NULL donne un résultat NULL.

---

# Fonctions de contrôle

Les fonctions suivantes sont présentées dans cette section :

`IF`

Fonction du type « si alors sinon » basée sur une condition.

`IFNULL`

Fonction du type « si alors sinon » basée sur la nullité d'une expression.

`NULLIF`

Retourne `NULL` si deux expressions sont égales.

`CASE`

Structure de contrôle condition du type « si alors sinon » (généralisation de la fonction `IF`).

## IF

*Syntaxe*

```
IF(condition,valeur_si_vrai,valeur_si_faux)
```

Si l'expression `condition` est vraie (`TRUE`), la fonction retourne l'expression `valeur_si_vrai` ; sinon (`condition = FALSE` ou `NULL`), elle retourne l'expression `valeur_si_faux`.

*Exemple*

```
mysql> SELECT
    ->   titre,
    ->   annee_parution,
    ->   IF(2008-annee_parution > 2,'Ancien','Récent') age
    -> FROM livre
    -> WHERE id_collection = 1;
+-------------------+----------------+--------+
| titre             | annee_parution | age    |
+-------------------+----------------+--------+
| PHP 5.2           |           2007 | Récent |
| Oracle 10g        |           2005 | Ancien |
| BusinessObjects 6 |           2004 | Ancien |
| MySQL 5           |           2006 | Récent |
+-------------------+----------------+--------+
4 rows in set (0.00 sec)
```

## IFNULL

*Syntaxe*

```
IFNULL(expression,valeur_si_null)
```

Si l'expression `expression` est `NULL`, la fonction retourne l'expression `valeur_si_null` ; sinon (`expression` n'est pas `NULL`), elle retourne l'expression `expression`.

La fonction `IFNULL` est très utile dans les expressions pour donner une valeur à une expression `NULL` et éviter d'avoir un résultat `NULL`.

*Exemple*

```
mysql> SELECT
    ->   nom,prix_ht,frais_ht,prix_ht+frais_ht total_ht
    -> FROM collection;
+--------------------------+---------+----------+----------+
| nom                      | prix_ht | frais_ht | total_ht |
```

```
+-------------------------+---------+----------+----------+
| Ressources Informatiques |   24.44 |     1.50 |    25.94 |
| TechNote                 |   10.48 |     NULL |     NULL |
| Les TP Informatiques     |   25.59 |     1.50 |    27.09 |
| Coffret Technique        |   46.45 |     2.00 |    48.45 |
| Solutions Informatiques  |   36.97 |     1.25 |    38.22 |
| ExpertIT                 |   20.00 |     NULL |     NULL |
+-------------------------+---------+----------+----------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->   nom,prix_ht,frais_ht,prix_ht+IFNULL(frais_ht,0) total_ht
    -> FROM collection;
+-------------------------+---------+----------+----------+
| nom                     | prix_ht | frais_ht | total_ht |
+-------------------------+---------+----------+----------+
| Ressources Informatiques |   24.44 |     1.50 |    25.94 |
| TechNote                 |   10.48 |     NULL |    10.48 |
| Les TP Informatiques     |   25.59 |     1.50 |    27.09 |
| Coffret Technique        |   46.45 |     2.00 |    48.45 |
| Solutions Informatiques  |   36.97 |     1.25 |    38.22 |
| ExpertIT                 |   20.00 |     NULL |    20.00 |
+-------------------------+---------+----------+----------+
6 rows in set (0.00 sec)
```

### NULLIF

*Syntaxe*

```
NULLIF(expression1,expression2)
```

Si les expressions expression1 et expression2 sont égales, la fonction retourne NULL ; sinon (expression n'est pas NULL), elle retourne l'expression expression1.

### CASE

*Syntaxe 1*

```
CASE valeur_a_tester
WHEN valeur1 THEN résultat1
[ WHEN valeur2 THEN résultat2 ]
[...]
[ ELSE resultat ]
END
```

*Syntaxe 2*

```
CASE
WHEN condition1 THEN résultat1
[ WHEN condition2 THEN résultat2 ]
[...]
[ ELSE resultat ]
END
```

Avec la première syntaxe, la fonction retourne la première expression résultatN pour laquelle les expressions valeur_a_tester et valeurN sont égales. Avec la deuxième syntaxe, la fonction retourne la première expression résultatN pour laquelle l'expression conditionN est vraie. Dans les deux syntaxes, s'il n'y a pas de correspondance, c'est l'expression resultat de la clause ELSE qui est retournée (NULL s'il n'y a pas de ELSE).

*Exemple*

```
mysql> SELECT
    ->   titre,
    ->   nombre_pages,
    ->   CASE
    ->     WHEN nombre_pages < 400
    ->     THEN 1.5
    ->     WHEN nombre_pages BETWEEN 400 AND 500
```

```
    ->     THEN 2
    ->     ELSE 2.5
    ->   END frais_envoi
    -> FROM livre
    -> WHERE id_collection <> 2
    -> ORDER BY nombre_pages;
+---------------------------------+--------------+-------------+
| titre                           | nombre_pages | frais_envoi |
+---------------------------------+--------------+-------------+
| PHP et MySQL (versions 4 et 5) |          272 |         1.5 |
| MySQL 5                         |          468 |           2 |
| BusinessObjects 6               |          470 |           2 |
| Oracle 10g                      |          489 |           2 |
| PHP 5.2                         |          518 |         2.5 |
| MySQL 5 et PHP 5                |          972 |         2.5 |
+---------------------------------+--------------+-------------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->   titre,
    ->   annee_parution,
    ->   CASE annee_parution
    ->     WHEN 2007 THEN 'Très récent'
    ->     WHEN 2006 THEN 'Récent'
    ->     ELSE           'Ancien'
    ->   END age
    -> FROM livre
    -> WHERE id_collection = 1;
+-------------------+----------------+-------------+
| titre             | annee_parution | age         |
+-------------------+----------------+-------------+
| PHP 5.2           |           2007 | Très récent |
| Oracle 10g        |           2005 | Ancien      |
| BusinessObjects 6 |           2004 | Ancien      |
| MySQL 5           |           2006 | Récent      |
+-------------------+----------------+-------------+
4 rows in set (0.00 sec)
```

# Fonctions de comparaison

Les fonctions suivantes sont présentées dans cette section :

LEAST

Plus petite valeur d'une liste de valeurs.

GREATEST

Plus grande valeur d'une liste de valeurs.

COALESCE

Première expression non NULL d'une liste d'expressions.

## LEAST - GREATEST

*Syntaxe*

```
LEAST(expression1,expression2[,...])
GREATEST(expression1,expression2[,...])
```

Les fonctions LEAST et GREATEST retournent respectivement la plus petite et la plus grande valeur d'une liste d'expressions.

*Exemple*

```
mysql> -- Calcul du montant d'une remise
mysql> -- de 5% plafonnée à 1.5
mysql> SELECT
    ->   nom,
    ->   prix_ht,
    ->   LEAST(ROUND(prix_ht*5/100,2),1.5) remise
    -> FROM collection;
+--------------------------+---------+--------+
| nom                      | prix_ht | remise |
+--------------------------+---------+--------+
| Ressources Informatiques |   24.44 |   1.22 |
| TechNote                 |   10.48 |   0.52 |
| Les TP Informatiques     |   25.59 |   1.28 |
| Coffret Technique        |   46.45 |    1.5 |
| Solutions Informatiques  |   36.97 |    1.5 |
| ExpertIT                 |   20.00 |   1.00 |
+--------------------------+---------+--------+
6 rows in set (0.01 sec)

mysql> -- Calcul du montant d'une remise
mysql> -- de 5% avec un minimum de 1
mysql> SELECT
    ->   nom,
    ->   prix_ht,
    ->   GREATEST(ROUND(prix_ht*5/100,2),1) remise
    -> FROM collection;
+--------------------------+---------+--------+
| nom                      | prix_ht | remise |
+--------------------------+---------+--------+
| Ressources Informatiques |   24.44 |   1.22 |
| TechNote                 |   10.48 |      1 |
| Les TP Informatiques     |   25.59 |   1.28 |
| Coffret Technique        |   46.45 |   2.32 |
| Solutions Informatiques  |   36.97 |   1.85 |
| ExpertIT                 |   20.00 |   1.00 |
+--------------------------+---------+--------+
6 rows in set (0.00 sec)
```

## COALESCE

*Syntaxe*

```
COALESCE(expression1,expression2[,...])
```

La fonction retourne la première expression `expressionN` qui est non `NULL` ; la fonction retourne `NULL` si toutes les expressions sont `NULL`.

*Exemple*

```
mysql> -- Afficher un numéro de téléphone parmi les numéros de
mysql> -- téléphones possibles des auteurs
mysql> SELECT
    ->   nom,
    ->   tel_bureau,
    ->   tel_portable,
    ->   tel_domicile,
    ->   COALESCE(tel_bureau,tel_portable,tel_domicile) telephone
    -> FROM
    ->   auteur;
+-----------+------------+--------------+--------------+------------+
| nom       | tel_bureau | tel_portable | tel_domicile | telephone  |
+-----------+------------+--------------+--------------+------------+
| HEURTEL   | NULL       | 0687731346   | 0102030405   | 0687731346 |
| THIBAUD   | 0203040506 | NULL         | NULL         | 0203040506 |
| GUERIN    | NULL       | NULL         | 0304050607   | 0304050607 |
| NOIRAULT  | NULL       | NULL         | NULL         | NULL       |
| GABILLAUD | NULL       | NULL         | NULL         | NULL       |
+-----------+------------+--------------+--------------+------------+
5 rows in set (0.01 sec)
```

# Fonctions numériques

Les fonctions suivantes sont présentées dans cette section :

`ABS`

Valeur absolue d'un nombre.

`CEILING`, `CEIL`

Plus petit entier qui n'est pas inférieur à un nombre.

`DIV`

Résultat de la division entière de deux nombres.

`FLOOR`

Plus grand entier qui n'est pas supérieur à un nombre.

`MOD`, `%`

Reste de la division entière de deux nombres.

`RAND`

Nombre aléatoire supérieur ou égal à 0 et strictement inférieur à 1.

`ROUND`

Nombre arrondi à la précision demandée.

`TRUNCATE`

Nombre tronqué à la précision demandée.

---

 Une division par zéro donne un résultat NULL.

---

## ABS

*Syntaxe*

```
ABS(nombre)
```

La fonction `ABS` retourne la valeur absolue d'un nombre.

## CEILING - CEIL

*Syntaxe*

```
CEIL(nombre)
```

La fonction `CEILING` (ou son synonyme `CEIL`) retourne le plus petit entier qui n'est pas inférieur à un nombre.

*Exemple*

```
mysql> SELECT nom,prix_ht,CEIL(prix_ht) FROM collection;
+---------------------------+---------+---------------+
| nom                       | prix_ht | CEIL(prix_ht) |
+---------------------------+---------+---------------+
| Ressources Informatiques  |   24.44 |            25 |
| TechNote                  |   10.48 |            11 |
| Les TP Informatiques      |   25.59 |            26 |
| Coffret Technique         |   46.45 |            47 |
```

```
| Solutions Informatiques  |   36.97 |             37 |
| ExpertIT                 |   20.00 |             20 |
+--------------------------+---------+---------------+
6 rows in set (0.00 sec)
```

## DIV

*Syntaxe*

```
dividende DIV diviseur
```

L'opérateur `DIV` retourne le résultat de la division entière de deux nombres.

*Exemple*

```
mysql> SELECT
    ->   titre,nombre_pages,nombre_pages DIV 100 nb_cent_pages
    -> FROM livre WHERE id_collection = 1;
+-------------------+--------------+---------------+
| titre             | nombre_pages | nb_cent_pages |
+-------------------+--------------+---------------+
| PHP 5.2           |          518 |             5 |
| Oracle 10g        |          489 |             4 |
| BusinessObjects 6 |          470 |             4 |
| MySQL 5           |          468 |             4 |
+-------------------+--------------+---------------+
4 rows in set (0.00 sec)
```

## FLOOR

*Syntaxe*

```
FLOOR(nombre)
```

La fonction `FLOOR` retourne le plus grand entier qui n'est pas supérieur à un nombre.

*Exemple*

```
mysql> SELECT nom,prix_ht,FLOOR(prix_ht) FROM collection;
+--------------------------+---------+----------------+
| nom                      | prix_ht | FLOOR(prix_ht) |
+--------------------------+---------+----------------+
| Ressources Informatiques |   24.44 |             24 |
| TechNote                 |   10.48 |             10 |
| Les TP Informatiques     |   25.59 |             25 |
| Coffret Technique        |   46.45 |             46 |
| Solutions Informatiques  |   36.97 |             36 |
| ExpertIT                 |   20.00 |             20 |
+--------------------------+---------+----------------+
6 rows in set (0.00 sec)
```

## MOD - %

*Syntaxe*

```
MOD(dividende,diviseur)
dividende MOD diviseur
dividende % diviseur
```

La fonction `MOD` (modulo) retourne le reste de la division entière de deux nombres. `MOD` est aussi un opérateur ; l'opérateur `%` donne le même résultat.

*Exemple*

```
mysql> SELECT
    ->   titre,nombre_pages,nombre_pages MOD 100 reste_pages
    -> FROM livre WHERE id_collection = 1;
```

```
+-------------------+-------------+-------------+
| titre             | nombre_pages | reste_pages |
+-------------------+-------------+-------------+
| PHP 5.2           |         518 |          18 |
| Oracle 10g        |         489 |          89 |
| BusinessObjects 6 |         470 |          70 |
| MySQL 5           |         468 |          68 |
+-------------------+-------------+-------------+
4 rows in set (0.01 sec)
```

## RAND

*Syntaxe*

```
RAND()
```

La fonction RAND retourne un nombre aléatoire supérieur ou égal à 0 et strictement inférieur à 1.

*Exemple*

```
mysql> SELECT rand(),rand(),rand();
+------------------+-------------------+------------------+
| rand()           | rand()            | rand()           |
+------------------+-------------------+------------------+
| 0.23858279198276 | 0.075853456813706 | 0.66351893885389 |
+------------------+-------------------+------------------+
1 row in set (0.00 sec)
```

Cette fonction peut être utilisée pour trier le résultat d'une requête selon un ordre aléatoire, ou tirer au sort une ou plusieurs lignes.

*Exemple*

```
mysql> SELECT nom FROM collection ORDER BY rand() LIMIT 1;
+-------------------+
| nom               |
+-------------------+
| Coffret Technique |
+-------------------+
1 row in set (0.00 sec)

mysql> SELECT nom FROM collection ORDER BY rand() LIMIT 1;
+----------+
| nom      |
+----------+
| TechNote |
+----------+
1 row in set (0.00 sec)
```

## ROUND

*Syntaxe*

```
ROUND(nombre[,précision])
```

La fonction ROUND retourne un nombre arrondi à la précision demandée.

précision est égal à 0 par défaut (arrondi à l'entier le plus proche). précision peut être négatif pour arrondir à un certain nombre de chiffres avant la virgule.

*Exemple*

```
mysql> SELECT
    ->   nom,
    ->   prix_ht,
    ->   ROUND(prix_ht) r1,
    ->   ROUND(prix_ht,1) r2,
    ->   ROUND(prix_ht,-1) r3
```

```
    -> FROM collection;
+--------------------------+---------+------+------+------+
| nom                      | prix_ht | r1   | r2   | r3   |
+--------------------------+---------+------+------+------+
| Ressources Informatiques |   24.44 |   24 | 24.4 |   20 |
| TechNote                 |   10.48 |   10 | 10.5 |   10 |
| Les TP Informatiques     |   25.59 |   26 | 25.6 |   30 |
| Coffret Technique        |   46.45 |   46 | 46.5 |   50 |
| Solutions Informatiques  |   36.97 |   37 | 37.0 |   40 |
| ExpertIT                 |   20.00 |   20 | 20.0 |   20 |
+--------------------------+---------+------+------+------+
6 rows in set (0.00 sec)
```

### TRUNCATE

*Syntaxe*

```
TRUNCATE(nombre,précision)
```

La fonction TRUNCATE retourne un nombre tronqué à la précision demandée.

Si précision est égal à 0 la troncature s'effectue à l'entier le plus proche. précision peut être négatif pour tronquer à un certain nombre de chiffres avant la virgule.

*Exemple*

```
mysql> SELECT
    ->   nom,
    ->   prix_ht,
    ->   TRUNCATE(prix_ht,0) r1,
    ->   TRUNCATE(prix_ht,1) r2,
    ->   TRUNCATE(prix_ht,-1) r3
    -> FROM collection;
+--------------------------+---------+------+------+------+
| nom                      | prix_ht | r1   | r2   | r3   |
+--------------------------+---------+------+------+------+
| Ressources Informatiques |   24.44 |   24 | 24.4 |   20 |
| TechNote                 |   10.48 |   10 | 10.4 |   10 |
| Les TP Informatiques     |   25.59 |   25 | 25.5 |   20 |
| Coffret Technique        |   46.45 |   46 | 46.4 |   40 |
| Solutions Informatiques  |   36.97 |   36 | 36.9 |   30 |
| ExpertIT                 |   20.00 |   20 | 20.0 |   20 |
+--------------------------+---------+------+------+------+
6 rows in set (0.00 sec)
```

# Fonctions caractères

Les fonctions suivantes sont présentées dans cette section :

`CONCAT, CONCAT_WS`

Concaténation de chaînes de caractères.

`INSTR`

Position de la première occurrence d'une chaîne à l'intérieure d'une autre chaîne.

`LEFT, RIGHT`

n premiers ou n dernier caractères d'une chaîne.

`LENGTH`

Longueur d'une chaîne.

`LOWER, UPPER`

Chaîne en minuscules ou en majuscules.

`LPAD, RPAD`

Chaîne complétée à gauche ou à droite par une séquence de caractères jusqu'à une certaine longueur.

`LTRIM, RTRIM, TRIM`

Suppression d'espace (ou d'autres caractères) en début ou en fin de chaîne.

`REPEAT, SPACE`

Chaîne construite en répétant une séquence de caractères un certain nombre de fois.

`REPLACE`

Remplacement de toutes les occurrences d'une chaîne par une autre.

`SUBSTRING, SUBSTR, SUBSTRING_INDEX`

Portion d'une chaîne.

---

 Rappel : seules les chaînes de caractères « binaires » sont sensibles à la casse.

---

## CONCAT - CONCAT_WS

*Syntaxe*

```
CONCAT(chaîne1,chaîne2[,...])
CONCAT_WS(séparateur,chaîne1,chaîne2[,...])
```

La fonction `CONCAT` retourne une chaîne de caractères qui concatène tous ses arguments.

La fonction `CONCAT_WS` est une variante de la fonction `CONCAT`. Le premier argument est une chaîne qui est utilisée comme séparateur dans la concaténation des autres arguments.

*Exemple*

```
mysql> SELECT CONCAT(prenom,' ',nom) FROM auteur;
+------------------------+
| CONCAT(prenom,' ',nom) |
+------------------------+
```

```
| Jérôme GABILLAUD        |
| Brice-Arnaud GUERIN     |
| Olivier HEURTEL         |
| Claire NOIRAULT         |
| Cyril THIBAUD           |
+-------------------------+
5 rows in set (0.00 sec)

mysql> SELECT CONCAT_WS(',',nom,prenom) FROM auteur;
+---------------------------+
| CONCAT_WS(',',nom,prenom) |
+---------------------------+
| GABILLAUD,Jérôme          |
| GUERIN,Brice-Arnaud       |
| HEURTEL,Olivier           |
| NOIRAULT,Claire           |
| THIBAUD,Cyril             |
+---------------------------+
5 rows in set (0.00 sec)
```

### INSTR

*Syntaxe*

```
INSTR(chaîne,chaîne_cherchée)
```

La fonction INSTR retourne la position de la première occurrence d'une chaîne à l'intérieure d'une autre chaîne.

La fonction retourne 0 si la chaîne recherchée n'est pas trouvée.

*Exemple*

```
mysql> SELECT nom,INSTR(nom,'Informatique') FROM collection;
+--------------------------+---------------------------+
| nom                      | INSTR(nom,'Informatique') |
+--------------------------+---------------------------+
| Coffret Technique        |                         0 |
| ExpertIT                 |                         0 |
| Les TP Informatiques     |                         8 |
| Ressources Informatiques |                        12 |
| Solutions Informatiques  |                        11 |
| TechNote                 |                         0 |
+--------------------------+---------------------------+
6 rows in set (0.01 sec)
```

### LEFT - RIGHT

*Syntaxe*

```
LEFT(chaîne,nombre_caractères)
RIGHT(chaîne,nombre_caractères)
```

Les fonctions LEFT et RIGHT retournent respectivement les *n* premiers ou les *n* dernier caractères d'une chaîne.

*Exemple*

```
mysql> SELECT nom,LEFT(nom,8),RIGHT(nom,8) FROM collection;
+--------------------------+-------------+--------------+
| nom                      | LEFT(nom,8) | RIGHT(nom,8) |
+--------------------------+-------------+--------------+
| Coffret Technique        | Coffret     | echnique     |
| ExpertIT                 | ExpertIT    | ExpertIT     |
| Les TP Informatiques     | Les TP I    | matiques     |
| Ressources Informatiques | Ressourc    | matiques     |
| Solutions Informatiques  | Solution    | matiques     |
| TechNote                 | TechNote    | TechNote     |
+--------------------------+-------------+--------------+
6 rows in set (0.01 sec)
```

## LENGTH

*Syntaxe*

```
LENGTH(chaîne)
```

La fonction LENGTH retourne la longueur d'une chaîne.

*Exemple*

```
mysql> SELECT nom,LENGTH(nom) FROM collection;
+--------------------------+-------------+
| nom                      | LENGTH(nom) |
+--------------------------+-------------+
| Coffret Technique        |          17 |
| ExpertIT                 |           8 |
| Les TP Informatiques     |          20 |
| Ressources Informatiques |          24 |
| Solutions Informatiques  |          23 |
| TechNote                 |           8 |
+--------------------------+-------------+
6 rows in set (0.00 sec)
```

## LOWER - UPPER

*Syntaxe*

```
LOWER(chaîne)
UPPER(chaîne)
```

Les fonctions LOWER et UPPER retournent une chaîne avec toutes les lettres respectivement en minuscules ou en majuscules.

*Exemple*

```
mysql> SELECT LOWER(nom),UPPER(prenom) FROM auteur;
+------------+---------------+
| LOWER(nom) | UPPER(prenom) |
+------------+---------------+
| gabillaud  | JÉRÔME        |
| guerin     | BRICE-ARNAUD  |
| heurtel    | OLIVIER       |
| noirault   | CLAIRE        |
| thibaud    | CYRIL         |
+------------+---------------+
5 rows in set (0.00 sec)
```

## LPAD - RPAD

*Syntaxe*

```
LPAD(chaîne,longueur,caractères)
RPAD(chaîne,longueur,caractères)
```

Les fonctions LPAD et RPAD retournent une chaîne complétée respectivement à gauche ou à droite par une séquence de caractères jusqu'à une certaine longueur.

*Exemple*

```
mysql> SELECT LPAD(nom,30,'.') FROM collection;
+--------------------------------+
| LPAD(nom,30,'.')               |
+--------------------------------+
| .............Coffret Technique |
| ......................ExpertIT |
| ..........Les TP Informatiques |
| ......Ressources Informatiques |
```

```
| .......Solutions Informatiques |
| ......................TechNote |
+--------------------------------+
6 rows in set (0.00 sec)

mysql> SELECT RPAD(nom,30,'.') FROM collection;
+--------------------------------+
| RPAD(nom,30,'.')               |
+--------------------------------+
| Coffret Technique............. |
| ExpertIT...................... |
| Les TP Informatiques.......... |
| Ressources Informatiques...... |
| Solutions Informatiques....... |
| TechNote...................... |
+--------------------------------+
6 rows in set (0.01 sec)
```

## LTRIM - RTRIM - TRIM

*Syntaxe*

```
LTRIM(chaîne)
RTRIM(chaîne)
TRIM([[BOTH | LEADING | TRAILING] [caractères] FROM] chaîne)
```

Les fonctions LTRIM et RTRIM retournent une chaîne après suppression des espaces situés respectivement au début ou à la fin de la chaîne.

La fonction TRIM est une généralisation des fonctions LTRIM et RTRIM qui permet notamment de spécifier la séquence de caractères à supprimer.

caractères spécifie la séquence de caractères à supprimer (espace par défaut). BOTH, LEADING et TRAILING indiquent que la suppression doit s'effectuer respectivement des deux côtés (par défaut), au début ou à la fin de la chaîne.

*Exemple*

```
mysql> SET @x='   abc   ';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
    ->   CONCAT('|',@x,'|') x,
    ->   CONCAT('|',LTRIM(@x),'|') "ltrim(x)",
    ->   CONCAT('|',RTRIM(@x),'|') "rtrim(x)",
    ->   CONCAT('|',TRIM(@x),'|') "trim(x)";
+-------------+----------+----------+---------+
| x           | ltrim(x) | rtrim(x) | trim(x) |
+-------------+----------+----------+---------+
| |   abc   | | |abc   | | |   abc| | |abc|   |
+-------------+----------+----------+---------+
1 row in set (0.00 sec)

mysql> SET @x='***abc***';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
    ->   @x x,
    ->   TRIM(BOTH '*' FROM @x) "trim(x)";
+-----------+---------+
| x         | trim(x) |
+-----------+---------+
| ***abc*** | abc     |
+-----------+---------+
1 row in set (0.00 sec)
```

## REPEAT - SPACE

*Syntaxe*

```
REPEAT(caractères,nombre_fois)
SPACE(nombre_fois)
```

La fonction REPEAT retourne une chaîne construite en répétant une séquence de caractères un certain nombre de fois.

La fonction SPACE est un cas particulier de REPEAT où la séquence de caractères est un espace.

*Exemple*

```
mysql> SELECT CONCAT(REPEAT('*',5),SPACE(5),REPEAT('*',5)) x;
+-----------------+
| x               |
+-----------------+
| *****     ***** |
+-----------------+
1 row in set (0.01 sec)
```

## REPLACE

*Syntaxe*

```
REPLACE(chaîne,chaîne_cherchée,chaîne_remplacement)
```

La fonction REPLACE retourne une chaîne après remplacement de toutes les occurrences d'une chaîne par une autre.

*Exemple*

```
mysql> SELECT nom,REPLACE(nom,'Informatiques','Techniques') nouveau
    -> FROM collection WHERE nom LIKE '%informatiques%';
+--------------------------+-----------------------+
| nom                      | nouveau               |
+--------------------------+-----------------------+
| Les TP Informatiques     | Les TP Techniques     |
| Ressources Informatiques | Ressources Techniques |
| Solutions Informatiques  | Solutions Techniques  |
+--------------------------+-----------------------+
3 rows in set (0.00 sec)
```

## SUBSTRING - SUBSTR - SUBSTRING_INDEX

*Syntaxe*

```
SUBSTRING(chaîne,position[,longueur])
SUBSTRING(chaîne FROM position [FOR longueur])
SUBSTRING_INDEX(chaîne,délimiteur,occurrence)
```

La fonction SUBSTRING retourne la portion d'une chaîne qui commence à une certaine position et qui a une certaine longueur. La fonction SUBSTR est un synonyme de la fonction SUBSTRING.

position spécifie la position du premier caractère à extraire (1 pour le premier caractère de la chaîne) ; si position est négatif, la position est comptée à partir de la fin de la chaîne.

longueur spécifie le nombre de caractères à extraire ; s'il est omis, la fonction retourne tous les caractères jusqu'à la fin de la chaîne.

La fonction SUBSTRING_INDEX retourne la portion d'une chaîne située avant ou après la nième occurrence d'un délimiteur.

délimiteur spécifie la chaîne utilisée comme délimiteur.

occurrence spécifie l'occurrence du délimiteur à utiliser (1 pour la première occurrence). Si occurrence est positif, les occurrences sont comptées en partant du début de la chaîne et la fonction retourne la portion de chaîne située avant le délimiteur. ; si occurrence est négatif, les occurrences sont comptées en partant de la fin de la chaîne et la fonction retourne la portion de chaîne située après le délimiteur.

*Exemple*

```
mysql> SELECT
    ->   nom,
    ->   SUBSTR(nom,6),
    ->   SUBSTR(nom,6,3)
```

```
    -> FROM collection;
+--------------------------+---------------------+------------------+
| nom                      | SUBSTR(nom,6)       | SUBSTR(nom,6,3) |
+--------------------------+---------------------+------------------+
| Coffret Technique        | et Technique        | et               |
| ExpertIT                 | tIT                 | tIT              |
| Les TP Informatiques     | P Informatiques     | P I              |
| Ressources Informatiques | urces Informatiques | urc              |
| Solutions Informatiques  | ions Informatiques  | ion              |
| TechNote                 | ote                 | ote              |
+--------------------------+---------------------+------------------+
6 rows in set (0.00 sec)

mysql> SELECT
    ->   nom,
    ->   SUBSTR(nom,-6),
    ->   SUBSTR(nom,-6,3)
    -> FROM collection;
+--------------------------+----------------+------------------+
| nom                      | SUBSTR(nom,-6) | SUBSTR(nom,-6,3) |
+--------------------------+----------------+------------------+
| Coffret Technique        | hnique         | hni              |
| ExpertIT                 | pertIT         | per              |
| Les TP Informatiques     | tiques         | tiq              |
| Ressources Informatiques | tiques         | tiq              |
| Solutions Informatiques  | tiques         | tiq              |
| TechNote                 | chNote         | chN              |
+--------------------------+----------------+------------------+
6 rows in set (0.00 sec)

mysql> SET @x='/chemin/vers/fichier.txt';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT SUBSTRING_INDEX(@x,'/',2);
+---------------------------+
| SUBSTRING_INDEX(@x,'/',2) |
+---------------------------+
| /chemin                   |
+---------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBSTRING_INDEX(@x,'/',-1);
+----------------------------+
| SUBSTRING_INDEX(@x,'/',-1) |
+----------------------------+
| fichier.txt                |
+----------------------------+
1 row in set (0.00 sec)
```

# Fonctions dates

Les fonctions suivantes sont présentées dans cette section :

ADDDATE, DATE_ADD, DATE_SUB, SUBDATE

Ajoute ou retranche un intervalle de temps à une date.

CURDATE, CURRENT_DATE, UTC_DATE

Date courante.

CURTIME, CURRENT_TIME, UTC_TIME

Heure courante.

CURRENT_TIMESTAMP, NOW, LOCALTIME, LOCALTIMESTAMP, SYSDATE, UTC_TIMESTAMP

Date/heure courante.

DATE

Extrait la partie date d'une date/heure.

DATEDIFF

Différence en nombre de jour entre deux dates.

DAYOFWEEK, WEEKDAY, DAYOFMONTH, DAYOFYEAR

Extrait le numéro du jour dans la semaine, dans le mois ou dans l'année d'une date.

EXTRACT

Extrait une composante d'une date.

LAST_DAY

Dernier jour du mois d'une date.

MONTH

Numéro de mois d'une date.

WEEK, WEEKOFYEAR

Numéro de semaine d'une date.

YEAR

Année d'une date.

---

À l'exception de SYSDATE, les fonctions qui retournent la date et/ou l'heure « courante » sont évaluées une fois au début de la requête ; ces fonctions retournent donc la date et/ou l'heure de début d'exécution de la requête. Si une telle fonction est appelée plusieurs fois à l'intérieur de la requête, c'est donc toujours la même valeur qui est retournée.

---

### ADDDATE - DATE_ADD - DATE_SUB - SUBDATE

*Syntaxe*

```
ADDDATE(date,INTERVAL valeur unité)
ADDDATE(date,nombre_jours)
DATE_ADD(date,INTERVAL valeur unité)
```

```
DATE_SUB(date,INTERVAL valeur unité)
SUBDATE(date,INTERVAL valeur unité)
SUBDATE(date,nombre_jours)
```

Les fonctions ADDDATE, DATE_ADD, DATE_SUB et SUBDATE retournent une date après ajout ou soustraction d'un intervalle de temps.

Dans les syntaxes avec le mot clé INTERVAL, unité est un mot clé qui donne l'unité de l'intervalle (voir ci-dessous) et valeur est une expression qui donne la valeur de l'intervalle à ajouter ou retrancher à la date (valeur peut être positif ou négatif).

*Exemples d'intervalle*

| **Mot clé** | **Valeur** |
| --- | --- |
| SECOND | Secondes |
| MINUTE | Minutes |
| HOUR | Heures |
| DAY | Jours |
| WEEK | Semaines |
| MONTH | Mois |
| QUARTER | Trimestres |
| YEAR | Années |
| MINUTE_SECOND | Minutes et secondes au format 'm:s' |
| HOUR_SECOND | Heures, minutes et secondes au format 'h:m:s' |
| HOUR_MINUTE | Heures et minutes au format 'h:m' |
| DAY_SECOND | Jours, heures, minutes et secondes au format 'j h:m:s' |
| DAY_MINUTE | Jours, heures et minutes au format 'j m:s' |

DAY_HOUR

Jours et heures au format 'j h'

YEAR_MONTH

Années et mois au format 'a-m'

---

Il existe aussi des intervalles qui permettent de spécifier des microsecondes. N'importe quel délimiteur peut être utilisé dans les formats.

---

ADDDATE, dans sa syntaxe avec le mot clé INTERVAL, est un synonyme de DATE_ADD. SUBDATE, dans sa syntaxe avec le mot clé INTERVAL, est un synonyme de DATE_SUB.

ADDDATE(date,n) est équivalent à DATE_ADD(date,INTERVAL n DAY). SUBDATE(date,n) est équivalent à DATE_SUB (date,INTERVAL n DAY).

Les mêmes opérations peuvent être effectuées directement sur une date en utilisant les opérateurs +, - et le mot clé INTERVAL :

```
date + INTERVAL valeur unité
```

```
date - INTERVAL valeur unité
```

---

Lors d'une opération avec les mois et les années, si le résultat donne un jour plus grand que le nombre de jours du mois, le jour est ajusté au nombre maximum de jours du mois résultat.

---

*Exemple*

```
mysql> SET @d='2008-01-22';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
    ->   @d,
    ->   ADDDATE(@d,1) "+ 1 jour",
    ->   ADDDATE(@d,INTERVAL 10 DAY)  "+ 10 jours",
    ->   ADDDATE(@d,INTERVAL '1-3' YEAR_MONTH)  "+ 1 an et 3 mois";
+------------+------------+------------+------------------+
| @d         | + 1 jour   | + 10 jours | + 1 an et 3 mois |
+------------+------------+------------+------------------+
| 2008-01-22 | 2008-01-23 | 2008-02-01 | 2009-04-22       |
+------------+------------+------------+------------------+
1 row in set (0.00 sec)
```

## CURDATE - CURRENT_DATE - UTC_DATE

*Syntaxe*

```
CURDATE()
CURRENT_DATE()
UTC_DATE()
```

Les fonctions CURDATE et CURRENT_DATE retournent la date du début d'exécution de la requête au format YYYY-MM-DD ou YYYYMMDD selon que la fonction est appelée dans un contexte de chaîne ou de nombre.

La fonction UTC_DATE retourne la même chose mais pour la date UTC.

*Exemple*

```
mysql> SELECT CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2008-01-22 |
+------------+
```

```
1 row in set (0.00 sec)

mysql> SELECT date_debut,date_fin FROM promotion WHERE id = 2;
+------------+----------+
| date_debut | date_fin |
+------------+----------+
| NULL       | NULL     |
+------------+----------+
1 row in set (0.01 sec)

mysql> UPDATE promotion
    -> SET date_debut = CURDATE(),date_fin = ADDDATE(CURDATE(),7)
    -> WHERE id = 2;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT date_debut,date_fin FROM promotion WHERE id = 2;
+------------+------------+
| date_debut | date_fin   |
+------------+------------+
| 2008-01-22 | 2008-01-29 |
+------------+------------+
1 row in set (0.00 sec)
```

## CURTIME - CURRENT_TIME - UTC_TIME

*Syntaxe*

```
CURTIME()
CURRENT_TIME()
UTC_TIME()
```

Les fonctions CURTIME et CURRENT_TIME retournent l'heure du début d'exécution de la requête au format HH:MM:SS ou HHMMSS selon que la fonction est appelée dans un contexte de chaîne ou de nombre.

La fonction UTC_TIME retourne la même chose mais pour l'heure UTC.

*Exemple*

```
mysql> SELECT CURTIME();
+-----------+
| CURTIME() |
+-----------+
| 15:30:16  |
+-----------+
1 row in set (0.00 sec)
```

## CURRENT_TIMESTAMP - NOW - LOCALTIME - LOCALTIMESTAMP - SYSDATE - UTC_TIMESTAMP

*Syntaxe*

```
CURRENT_TIMESTAMP()
NOW()
LOCALTIME()
LOCALTIMESTAMP()
SYSDATE()
UTC_TIMESTAMP()
```

La fonction NOW retourne la date/heure du début d'exécution de la requête au format YYYY-MM-DD HH:MM:SS ou YYYYMMDDHHMMSS selon que la fonction est appelée dans un contexte de chaîne ou de nombre. Les fonctions CURRENT_TIMESTAMP, LOCALTIME et LOCALTIMESTAMP sont des synonymes de la fonction NOW. Si la fonction NOW est appelée plusieurs fois dans une requête (ou une procédure stockée), elle retourne toujours le même résultat.

La fonction UTC_TIMESTAMP retourne la même chose mais pour la date/heure UTC.

La fonction SYSDATE retourne la date/heure courante au format YYYY-MM-DD HH:MM:SS ou YYYYMMDDHHMMSS selon que la fonction est appelée dans un contexte de chaîne ou de nombre. Si la fonction SYSDATE est appelée plusieurs fois dans une requête (ou une procédure stockée), elle est réévaluée à chaque fois.

*Exemple*

```
mysql> SELECT NOW(),UTC_TIMESTAMP();
+---------------------+---------------------+
| NOW()               | UTC_TIMESTAMP()     |
+---------------------+---------------------+
| 2008-01-22 15:30:51 | 2008-01-22 14:30:51 |
+---------------------+---------------------+
1 row in set (0.00 sec)
```

## DATE

*Syntaxe*

```
DATE(date)
```

La fonction DATE retourne la partie date d'une date/heure.

*Exemple*

```
mysql> SELECT date_maj,DATE(date_maj) FROM livre WHERE id = 2;
+---------------------+----------------+
| date_maj            | DATE(date_maj) |
+---------------------+----------------+
| 2008-01-22 15:26:16 | 2008-01-22     |
+---------------------+----------------+
1 row in set (0.00 sec)
```

## DATEDIFF

*Syntaxe*

```
DATEDIFF(date1,date2)
```

La fonction DATEDIFF retourne la différence en nombre de jours entre deux dates (les composantes horaires éventuelles des deux dates sont ignorées).

*Exemple*

```
mysql> SELECT DATEDIFF(date_fin,date_debut)
    -> FROM promotion WHERE id = 2;
+-------------------------------+
| DATEDIFF(date_fin,date_debut) |
+-------------------------------+
|                             7 |
+-------------------------------+
1 row in set (0.00 sec)
```

## DAYOFWEEK - DAYOFMONTH - DAYOFYEAR - WEEKDAY

*Syntaxe*

```
DAYOFWEEK(date)
DAYOFMONTH(date)
DAYOFYEAR(date)
```

Les fonctions DAYOFWEEK (ou WEEKDAY), DAYOFMONTH et DAYOFYEAR retournent respectivement le numéro du jour dans la semaine, dans le mois ou dans l'année d'une date.

La fonction DAYOFWEEK retourne un nombre compris entre 1 et 7 (1 = dimanche). La fonction WEEKDAY retourne un nombre compris entre 0 et 6 (0 = lundi).

*Exemple*

```
mysql> SET @d='2008-04-06'; -- dimanche 6 avril
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT
    ->   @d,
    ->   DAYOFWEEK(@d),
    ->   DAYOFMONTH(@d),
    ->   DAYOFYEAR(@d)
    ->   ;
+------------+---------------+----------------+---------------+
| @d         | DAYOFWEEK(@d) | DAYOFMONTH(@d) | DAYOFYEAR(@d) |
+------------+---------------+----------------+---------------+
| 2008-04-06 |             1 |              6 |            97 |
+------------+---------------+----------------+---------------+
1 row in set (0.00 sec)
```

## EXTRACT

*Syntaxe*

```
EXTRACT(unité FROM date)
```

La fonction EXTRACT retourne une composante d'une date.

unité est un mot clé qui spécifie la composante à extraire ; ce sont les mêmes mots clés que pour la spécification de l'intervalle dans les fonctions DATE_ADD et DATE_SUB.

*Exemple*

```
mysql> SELECT
    ->   NOW(),
    ->   EXTRACT(DAY FROM NOW()) jour,
    ->   EXTRACT(MONTH FROM NOW()) mois,
    ->   EXTRACT(YEAR FROM NOW()) annee,
    ->   EXTRACT(HOUR FROM NOW()) heure
    ->   ;
+---------------------+------+------+-------+-------+
| NOW()               | jour | mois | annee | heure |
+---------------------+------+------+-------+-------+
| 2008-01-22 15:42:25 |   22 |    1 |  2008 |    15 |
+---------------------+------+------+-------+-------+
1 row in set (0.00 sec)
```

## LASTDAY

*Syntaxe*

```
LAST_DAY(date)
```

La fonction LAST_DAY retourne une date correspondant au dernier jour du mois d'une date.

*Exemple*

```
mysql> SELECT
    ->   LAST_DAY(NOW()),
    ->   LAST_DAY(NOW()+INTERVAL 1 MONTH);
+-----------------+----------------------------------+
| LAST_DAY(NOW()) | LAST_DAY(NOW()+INTERVAL 1 MONTH) |
+-----------------+----------------------------------+
| 2008-01-31      | 2008-02-29                       |
+-----------------+----------------------------------+
1 row in set (0.01 sec)
```

## MONTH

*Syntaxe*

```
MONTH(date)
```

La fonction MONTH retourne le numéro de mois d'une date.

*Exemple*

```
mysql> SELECT CURRENT_DATE(),MONTH(CURRENT_DATE());
+----------------+-----------------------+
| CURRENT_DATE() | MONTH(CURRENT_DATE()) |
+----------------+-----------------------+
| 2008-01-22     |                     1 |
+----------------+-----------------------+
1 row in set (0.00 sec)
```

## WEEK - WEEKOFYEAR

*Syntaxe*

```
WEEK(date,mode)
WEEKOFYEAR(date)
```

Les fonctions `WEEK` et `WEEKOFYEAR` retournent le numéro de semaine d'une date.

`mode` permet d'indiquer si la semaine démarre un dimanche ou un lundi, et si le résultat doit être compris entre 0 et 53 ou entre 1 et 53. Les valeurs possibles sont les suivantes :

| **Mode** | **Premier jour de la semaine** | **Résultat** | **La semaine 1 est la semaine** |
|---|---|---|---|
| 0 | Dimanche | 0-53 | Avec un dimanche dans cette année |
| 1 | Lundi | 0-53 | Avec plus de 3 jours cette année |
| 2 | Dimanche | 1-53 | Avec un dimanche dans cette année |
| 3 | Lundi | 1-53 | Avec plus de 3 jours cette année |
| 4 | Dimanche | 0-53 | Avec plus de 3 jours cette année |
| 5 | Lundi | 0-53 | Avec un lundi dans cette année |
| 6 | Dimanche | 1-53 | Avec plus de 3 jours cette année |
| 7 | Lundi | 1-53 | Avec un lundi dans cette année |

`WEEKOFYEAR(date)` est équivalent `WEEK(date,3)`.

*Exemple*

```
mysql> SELECT CURRENT_DATE(),WEEKOFYEAR(CURRENT_DATE());
+----------------+----------------------------+
| CURRENT_DATE() | WEEKOFYEAR(CURRENT_DATE()) |
+----------------+----------------------------+
| 2008-01-22     |                          4 |
+----------------+----------------------------+
1 row in set (0.00 sec)
```

## YEAR

*Syntaxe*

```
YEAR(date)
```

La fonction YEAR retourne l'année d'une date.

*Exemple*

```
mysql> SELECT CURRENT_DATE(),YEAR(CURRENT_DATE());
+----------------+----------------------+
| CURRENT_DATE() | YEAR(CURRENT_DATE()) |
+----------------+----------------------+
| 2008-01-22     |                 2008 |
+----------------+----------------------+
1 row in set (0.01 sec)
```

# Fonctions de transtypage et de mise en forme

Les fonctions suivantes sont présentées dans cette section :

`BINARY`

Conversion d'une chaîne en chaîne binaire.

`CAST`, `CONVERT`

Conversion d'une donnée d'un type en un autre.

`DATE_FORMAT`

Formate une date.

`FORMAT`

Formate un nombre.

`STR_TO_DATE`

Conversion d'une chaîne en date.

## BINARY

*Syntaxe*

```
BINARY chaîne
```

L'opérateur `BINARY` convertit une chaîne en chaîne binaire.

*Exemple*

```
mysql> -- Recherche non sensible à la casse
mysql> SELECT prix_ht FROM collection WHERE nom = 'TECHNOTE';
+---------+
| prix_ht |
+---------+
|   10.48 |
+---------+
1 row in set (0.00 sec)

mysql> -- Recherche sensible à la casse
mysql> SELECT prix_ht FROM collection WHERE nom = BINARY 'TECHNOTE';
Empty set (0.00 sec)
```

## CAST - CONVERT

*Syntaxe*

```
CAST(expression AS type)
CONVERT(expression,type)
```

Les fonctions `CAST` et `CONVERT` convertissent une expression d'un type quelconque dans un autre type.

`type` peut être une des valeurs suivantes :

`BINARY[(n)]`

Chaîne binaire (éventuellement limitée à `n` octets)

`CHAR[(n)]`

Chaîne binaire (éventuellement limitée à `n` caractères)

DATE

Date

DATETIME

Date/heure

DECIMAL

Nombre décimal

SIGNED [INTEGER]

Entier signé

TIME

Heure

UNSIGNED [INTEGER]

Entier non signé

*Exemple*

```
mysql> SELECT
    ->   prix_ht,
    ->   CAST(prix_ht AS SIGNED) entier,
    ->   CAST(prix_ht AS CHAR) chaine
    -> FROM
    ->   collection;
+---------+--------+--------+
| prix_ht | entier | chaine |
+---------+--------+--------+
|   24.44 |     24 | 24.44  |
|   10.48 |     10 | 10.48  |
|   25.59 |     26 | 25.59  |
|   46.45 |     46 | 46.45  |
|   36.97 |     37 | 36.97  |
|   20.00 |     20 | 20.00  |
+---------+--------+--------+
6 rows in set (0.00 sec)
```

Si la conversion ne peut pas être effectuée, la fonction retourne une valeur arbitraire mais ne génère pas d'erreur.

## DATE_FORMAT

*Syntaxe*

```
DATE_FORMAT(date,format)
```

La fonction DATE_FORMAT retourne une date formatée selon un certain format.

format permet de spécifier le format ; les séquences suivantes peuvent être utilisées pour spécifier le format (liste non exhaustive) :

%e

Numéro du jour du mois (1-31)

%d

Numéro du jour du mois (01-31)

%c

Numéro du mois (1-12)

%m

Numéro du mois (01-12)

%Y

Année sur 4 chiffres

%y

Année sur 2 chiffres

%H

Heure (00-23)

%i

Minutes (00-59)

%S, %s

Secondes (00-59)

%T

Heure (sur 24 heures), minutes et secondes au format hh:mm:ss

%j

Numéro du jour dans l'année (001-366)

%W

Nom du jour de la semaine

%M

Nom du mois

La variable système lc_time_names définit la langue utilisée pour afficher les noms de jour ou de mois. La valeur fr_FR peut être utilisée pour avoir des noms en français (SET lc_time_names = 'fr_FR').

La fonction GET_FORMAT peut être utilisée pour récupérer des formats « standards ».

*Syntaxe* :

```
GET_FORMAT(DATE | TIME | TIMESTAMP, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')
```

| **Appel** | **Résultat** |
|---|---|
| GET_FORMAT(DATE,'USA') | '%m.%d.%Y' |
| GET_FORMAT(DATE,'JIS') | '%Y-%m-%d' |
| GET_FORMAT(DATE,'ISO') | '%Y-%m-%d' |
| GET_FORMAT(DATE,'EUR') | '%d.%m.%Y' |
| GET_FORMAT(DATE,'INTERNAL') | '%Y%m%d' |

| | |
|---|---|
| GET_FORMAT(TIMESTAMP,'USA') | '%Y-%m-%d-%H.%i.%s' |
| GET_FORMAT(TIMESTAMP,'JIS') | '%Y-%m-%d %H:%i:%s' |
| GET_FORMAT(TIMESTAMP,'ISO') | '%Y-%m-%d %H:%i:%s' |
| GET_FORMAT(TIMESTAMP,'EUR') | '%Y-%m-%d-%H.%i.%s' |
| GET_FORMAT(TIMESTAMP,'INTERNAL') | '%Y%m%d%H%i%s' |
| GET_FORMAT(TIME,'USA') | '%h:%i:%s %p' |
| GET_FORMAT(TIME,'JIS') | '%H:%i:%s' |
| GET_FORMAT(TIME,'ISO') | '%H:%i:%s' |
| GET_FORMAT(TIME,'EUR') | '%H.%i.%S' |
| GET_FORMAT(TIME,'INTERNAL') | '%H%i%s' |

*Exemple*

```
mysql> SELECT
    ->   NOW(),
    ->   DATE_FORMAT(NOW(),'%d/%m/%Y %H:%i:%s') france,
    ->   DATE_FORMAT(NOW(),GET_FORMAT(TIMESTAMP,'EUR')) europe;
+---------------------+---------------------+---------------------+
| NOW()               | france              | europe              |
+---------------------+---------------------+---------------------+
| 2008-01-22 16:21:50 | 22/01/2008 16:21:50 | 2008-01-22 16.21.50 |
+---------------------+---------------------+---------------------+
1 row in set (0.00 sec)
```

## FORMAT

*Syntaxe*

```
FORMAT(nombre,décimales)
```

La fonction FORMAT retourne un nombre avec une mise en forme du type #,###,###.##.

Le nombre est arrondi au nombre de décimales spécifié par l'argument décimales.

*Exemple*

```
mysql> SET @x=1234.567;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x,FORMAT(@x,2);
+----------+-------------+
| @x       | FORMAT(@x,2) |
+----------+-------------+
| 1234.567 | 1,234.57     |
+----------+-------------+
1 row in set (0.00 sec)
```

## STR_TO_DATE

*Syntaxe*

```
STR_TO_DATE(chaîne,format)
```

La fonction STR_TO_DATE convertit une chaîne en date ; c'est l'inverse de la fonction FORMAT_DATE.

format permet de spécifier le format de la chaîne, à l'aide des mêmes symboles que ceux utilisés dans la fonction FORMAT_DATE.

*Exemple*

```
mysql> SET @x='20080122-180709';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x,STR_TO_DATE(@x,'%Y%m%d-%H%i%s');
+-----------------+--------------------------------+
| @x              | STR_TO_DATE(@x,'%Y%m%d-%H%i%s') |
+-----------------+--------------------------------+
| 20080122-180709 | 2008-01-22 18:07:09            |
+-----------------+--------------------------------+
1 row in set (0.00 sec)
```

# Fonctions système

Les fonctions suivantes sont présentées dans cette section :

CURRENT_USER, SESSION_USER, SYSTEM_USER, USER

Utilisateur courant.

DATABASE, SCHEMA

Base de données courante.

FOUND_ROWS

Nombre de lignes retournées par le dernier ordre SELECT.

LAST_INSERT_ID

Valeur automatiquement générée par une colonne de type AUTO_INCREMENT lors du dernier INSERT.

ROW_COUNT

Nombre de lignes mises à jour par le dernier ordre INSERT, UPDATE ou DELETE.

VERSION

Version de MySQL.

## CURRENT_USER - SESSION_USER - SYSTEM_USER - USER

*Syntaxe*

```
CURRENT_USER()
USER()
SESSION_USER()
SYSTEM_USER()
```

La fonction CURRENT_USER retourne le nom d'utilisateur et le nom de la machine de la session courante, sous la forme utilisateur@machine.

La fonction USER retourne le nom d'utilisateur et le nom de la machine spécifiés lors de l'identification avec le serveur MySQL, sous la forme utilisateur@machine. Les fonctions SESSION_USER et SYSTEM_USER sont des synonymes de la fonction USER.

Le résultat des deux fonctions peut être différent. Par exemple, si un client a été identifié par le serveur comme utilisateur anonyme, la fonction CURRENT_USER retournera un nom d'utilisateur vide, alors que la fonction USER retourne le nom réellement spécifié dans la chaîne de connexion.

*Exemple*

```
[root@xampp ~]# mysql -u root
...
mysql> SELECT CURRENT_USER(),USER();
+----------------+----------------+
| CURRENT_USER() | USER()         |
+----------------+----------------+
| root@localhost | root@localhost |
+----------------+----------------+
1 row in set (0.00 sec)

mysql> exit
Bye

[root@xampp ~]# mysql -u eni
...
mysql> SELECT CURRENT_USER(),USER();
+----------------+---------------+
```

```
| CURRENT_USER() | USER()        |
+----------------+---------------+
| @localhost     | eni@localhost |
+----------------+---------------+
1 row in set (0.00 sec)
```

## DATABASE - SCHEMA

*Syntaxe*

```
DATABASE()SCHEMA()
```

Les fonctions DATABASE et SCHEMA retournent le nom de la base de données courante (NULL s'il n'y a pas de base de données courante). La fonction SCHEMA est un synonyme de la fonction DATABASE qui est apparu en version 5.0.2.

*Exemple*

```
mysql> SELECT DATABASE(),SCHEMA();
+------------+----------+
| DATABASE() | SCHEMA() |
+------------+----------+
| NULL       | NULL     |
+------------+----------+
1 row in set (0.00 sec)

mysql> USE eni;
Database changed
mysql> SELECT DATABASE(),SCHEMA();
+------------+----------+
| DATABASE() | SCHEMA() |
+------------+----------+
| eni        | eni      |
+------------+----------+
1 row in set (0.00 sec)
```

## FOUND_ROWS

*Syntaxe*

```
FOUND_ROWS()
```

La fonction FOUND_ROWS retourne le nombre de lignes renvoyées par le dernier ordre SELECT.

*Exemple*

```
mysql> SELECT titre FROM livre WHERE id_collection = 1;
+-------------------+
| titre             |
+-------------------+
| PHP 5.2           |
| Oracle 10g        |
| BusinessObjects 6 |
| MySQL 5           |
+-------------------+
4 rows in set (0.00 sec)

mysql> SELECT FOUND_ROWS();
+--------------+
| FOUND_ROWS() |
+--------------+
|            4 |
+--------------+
1 row in set (0.01 sec)
```

## LAST_INSERT_ID

*Syntaxe*

```
LAST_INSERT_ID()
```

La fonction LAST_INSERT_ID retourne la valeur automatiquement générée par une colonne de type AUTO_INCREMENT lors du dernier INSERT. Si le dernier INSERT a inséré plusieurs lignes, c'est la valeur générée pour la première ligne insérée qui est retournée.

Lorsque vous utilisez un ordre INSERT IGNORE et qu'une ligne est ignorée, le compteur AUTO_INCREMENT est malgré tout incrémenté. Avant la version 5.1.12, la fonction LAST_INSERT_ID retourne la valeur générée pour la première ligne insérée, même si celle-ci a été ignorée ; depuis la version 5.1.12, la fonction retourne la valeur générée pour la première ligne insérée avec succès.

*Exemple*

```
mysql> INSERT INTO rubrique(titre) VALUES('Certification');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|               16 |
+------------------+
1 row in set (0.01 sec)
```

## ROW_COUNT

*Syntaxe*

```
ROW_COUNT()
```

La fonction ROW_COUNT retourne le nombre de lignes traitées avec succès par le dernier ordre INSERT, UPDATE ou DELETE.

*Exemple*

```
mysql> DELETE FROM catalogue;
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
1 row in set (0.01 sec)
```

## VERSION

*Syntaxe*

```
VERSION()
```

La fonction VERSION retourne la version du serveur MySQL.

*Exemple*

```
mysql> SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 5.0.45    |
+-----------+
1 row in set (0.00 sec)
```

# Fonctions de chiffrement et de compression

Les fonctions suivantes sont présentées dans cette section :

AES_ENCRYPT, AES_DECRYPT

Chiffrement/déchiffrement de données utilisant l'algorithme AES.

COMPRESS, UNCOMPRESS

Compression/décompression de données.

MD5, SHA1, SHA

Somme de vérification d'une chaîne.

PASSWORD

Mot de passe chiffré.

---

Les fonctions de chiffrage et de compression retournent des chaînes binaires ; pour le stockage en base de telles données, il est conseillé d'utiliser une colonne de type BLOB.

---

## AES_ENCRYPT - AES_DECRYPT

*Syntaxe*

```
AES_ENCRYPT(chaîne,clé)
AES_DECRYPT(chaîne,clé)
```

Les fonctions AES_ENCRYPT et AES_DECRYPT chiffrent et déchiffrent une chaîne en utilisant l'algorithme AES (*Advanced Encryption Standard*) avec une clé de 128 bits.

*Exemple*

```
mysql> UPDATE auteur
    -> SET mot_de_passe = AES_ENCRYPT('abc123','secretdefense')
    -> WHERE id = 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT mot_de_passe FROM auteur WHERE id = 1;
+------------------+
| mot_de_passe     |
+------------------+
| ~YU®(@zôB
Á« |
+------------------+
1 row in set (0.00 sec)

mysql> SELECT AES_DECRYPT(mot_de_passe,'secretdefense')
    -> FROM auteur WHERE id = 1;
+-------------------------------------------+
| AES_DECRYPT(mot_de_passe,'secretdefense') |
+-------------------------------------------+
| abc123                                    |
+-------------------------------------------+
1 row in set (0.00 sec)
```

## COMPRESS - UNCOMPRESS

*Syntaxe*

```
COMPRESS(chaîne)
UNCOMPRESS(chaîne)
```

Les fonctions COMPRESS et UNCOMPRESS compressent et décompressent une chaîne. Ces fonctions nécessitent que MySQL ait été compilé avec une librairie de compression ; si ce n'est pas le cas, ces fonctions retournent toujours NULL.

*Exemple*

```
mysql> UPDATE auteur
    -> SET profil = COMPRESS('Une longue description ...')
    -> WHERE id = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT profil FROM auteur WHERE id = 1;
+------------------------------------------+
| profil                                   |
+------------------------------------------+
|    x
      ÍKUÈÉÏK/MUHI-N.Ê,(ÉÌÏSĐÓÓ         A |
+------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT UNCOMPRESS(profil) FROM auteur WHERE id = 1;
+----------------------------+
| UNCOMPRESS(profil)         |
+----------------------------+
| Une longue description ... |
+----------------------------+
1 row in set (0.00 sec)
```

## MD5 - SHA1 - SHA

*Syntaxe*

```
MD5(chaîne)
SHA1(chaîne)
```

Les fonctions MD5 et SHA1 retournent la somme de vérification d'une chaîne en utilisant respectivement les algorithmes MD5 (128 bits) et SHA-1 (160 bits). La fonction SHA est un synonyme de la fonction SHA1.

*Exemple*

```
mysql> SELECT SHA1('Olivier Heurtel');
+------------------------------------------+
| SHA1('Olivier Heurtel')                  |
+------------------------------------------+
| 7c9e072cdc1132ce6e3746911cfc65d63db99959 |
+------------------------------------------+
1 row in set (0.00 sec)
```

## PASSWORD

*Syntaxe*

```
PASSWORD(chaîne)
```

La fonction PASSWORD est la fonction utilisée par MySQL pour chiffrer un mot de passe ; le chiffrage est irréversible.

*Exemple*

```
mysql> UPDATE auteur
    -> SET mot_de_passe = PASSWORD('abc123')
    -> WHERE id = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT 'OK' FROM auteur
    -> WHERE id = 1 AND mot_de_passe = PASSWORD('123abc');
Empty set (0.00 sec)
```

```
mysql> SELECT ’OK’ FROM auteur
    -> WHERE id = 1 AND mot_de_passe = PASSWORD(’abc123’);
+----+
| OK |
+----+
| OK |
+----+
1 row in set (0.00 sec)
```

# Fonctions d'agrégat

Les fonctions d'agrégat sont particulières : elles retournent une ligne de résultat par groupe de lignes en entrée.

Ces fonctions sont la plupart du temps utilisées dans les requêtes qui groupent les données (utilisation de la clause `GROUP BY`, cf. chapitre Techniques avancées avec MySQL - Grouper les données).

Si ces fonctions sont utilisées dans une requête qui n'effectue pas de groupement de données, cela revient à grouper toutes les lignes : la fonction retourne une seule ligne de résultat. Dans ce cas, la clause `SELECT` de la requête ne doit contenir que des expressions qui utilisent une fonction d'agrégat.

Les fonctions suivantes sont présentées dans cette section :

`MIN`, `MAX`

Minimum ou maximum.

`SUM`

Somme.

`AVG`

Moyenne.

`COUNT`

Nombre.

---

 Pour toutes ces fonctions, les valeurs `NULL` sont ignorées ; la présence d'une valeur `NULL` dans le calcul ne donne pas un résultat `NULL`.

---

## MIN - MAX

*Syntaxe*

```
MIN(expression)
MAX(expression)
```

Les fonctions `MIN` et `MAX` retournent respectivement le minimum et le maximum de toutes les valeurs de `expression`.

*Exemple*

```
mysql> SELECT MIN(nombre_pages),MAX(nombre_pages)
    -> FROM livre WHERE id_collection = 1;
+-------------------+-------------------+
| MIN(nombre_pages) | MAX(nombre_pages) |
+-------------------+-------------------+
|               447 |               518 |
+-------------------+-------------------+
1 row in set (0.00 sec)
```

## SUM - AVG

*Syntaxe*

```
SUM(expression)
AVG(expression)
```

Les fonctions `SUM` et `AVG` retournent respectivement la somme et la moyenne de toutes les valeurs de `expression`.

Pour la fonction `AVG`, le fait que les valeurs `NULL` soient ignorées influe sur le nombre de valeurs prises en compte dans le calcul.

*Exemple*

```
mysql> SELECT frais_ht FROM collection;
+----------+
| frais_ht |
+----------+
|     1.50 |
|     NULL |
|     1.50 |
|     2.00 |
|     1.25 |
|     NULL |
+----------+
6 rows in set (0.00 sec)

mysql> SELECT SUM(frais_ht),AVG(frais_ht) FROM collection;
+---------------+---------------+
| SUM(frais_ht) | AVG(frais_ht) |
+---------------+---------------+
|          6.25 |      1.562500 |
+---------------+---------------+
1 row in set (0.01 sec)

mysql> SELECT SUM(frais_ht),AVG(frais_ht),AVG(IFNULL(frais_ht,0))
    -> FROM collection;
+---------------+---------------+-------------------------+
| SUM(frais_ht) | AVG(frais_ht) | AVG(IFNULL(frais_ht,0)) |
+---------------+---------------+-------------------------+
|          6.25 |      1.562500 |                1.041667 |
+---------------+---------------+-------------------------+
1 row in set (0.00 sec)

mysql> -- Vérification
mysql> SELECT (1.5+1.5+2+1.25)/4,(1.5+0+1.5+2+1.25+0)/6;
+--------------------+------------------------+
| (1.5+1.5+2+1.25)/4 | (1.5+0+1.5+2+1.25+0)/6 |
+--------------------+------------------------+
|           1.562500 |               1.041667 |
+--------------------+------------------------+
1 row in set (0.00 sec)
```

### COUNT

*Syntaxe*

```
COUNT([DISTINCT] expression)
COUNT(*)
```

Dans sa première syntaxe, la fonction `COUNT` compte le nombre de fois où `expression` n'est pas `NULL`. Avec le mot clé `DISTINCT`, la fonction compte le nombre de valeurs distinctes de `expression`, en ignorant la valeur `NULL`.

Dans sa deuxième syntaxe, la fonction `COUNT` compte le nombre total de lignes.

*Exemple*

```
mysql> SELECT frais_ht FROM collection;
+----------+
| frais_ht |
+----------+
|     1.50 |
|     NULL |
|     1.50 |
|     2.00 |
|     1.25 |
|     NULL |
+----------+
6 rows in set (0.01 sec)

mysql> SELECT
    ->   COUNT(*),
    ->   COUNT(frais_ht),
```

```
    ->   COUNT(DISTINCT frais_ht)
    -> FROM collection;
+----------+-----------------+--------------------------+
| COUNT(*) | COUNT(frais_ht) | COUNT(DISTINCT frais_ht) |
+----------+-----------------+--------------------------+
|        6 |               4 |                        3 |
+----------+-----------------+--------------------------+
1 row in set (0.00 sec)
```

# Créer et supprimer une base de données

L'ordre SQL CREATE DATABASE permet de créer une nouvelle base de données.

*Syntaxe*

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nom_base
```

nom_base est le nom de la nouvelle base de données. Ce nom doit respecter les règles de nommage des objets MySQL.

CREATE SCHEMA est un synonyme de CREATE DATABASE (depuis la version 5, une base de données est aussi appelée "schéma").

Une erreur se produit si une base de données de même nom existe déjà et que la clause IF NOT EXISTS n'est pas présente.

Pour créer une base de données, il faut le privilège global CREATE.

Physiquement, une base de données MySQL se matérialise par un répertoire qui contiendra les fichiers correspondant aux différentes tables de la base de données.

*Exemple*

```
mysql> CREATE DATABASE biblio;
Query OK, 1 row affected (0.00 sec)
```

L'ordre SQL DROP DATABASE permet de supprimer une base de données.

*Syntaxe*

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nom_base
```

DROP SCHEMA est un synonyme de DROP DATABASE.

Une erreur se produit si la base de données n'existe pas et que la clause IF EXISTS n'est pas présente.

Pour supprimer une base de données, il faut le privilège global DROP.

 L'ordre DROP DATABASE supprime tout, sans demander de confirmation. Il faut y réfléchir à deux fois avant d'exécuter cette commande !

# Gérer les utilisateurs et les droits

## 1. Vue d'ensemble

Lors de l'installation de MySQL, un compte super-utilisateur nommé `root` est automatiquement créé.

Le compte `root` est normalement réservé à l'administration du serveur MySQL.

En complément du compte `root`, il est donc conseillé de créer au minimum un compte par application et éventuellement un compte par utilisateur final de l'application. De cette manière, il sera possible de gérer très finement les droits attribués à chaque utilisateur/application et de limiter les risques liés à l'utilisation du compte `root`.

Dans MySQL, un utilisateur est identifié de manière unique par la combinaison de deux informations :

- un nom d'utilisateur ;
- un nom d'hôte (ou adresse IP) à partir duquel l'utilisateur peut se connecter.

Chaque couple utilisateur/hôte est considéré par MySQL comme un utilisateur unique qui a un mot de passe pour se connecter (éventuellement aucun) et des droits. Un même utilisateur (au sens d'un nom d'utilisateur donné) peut donc avoir des droits différents selon l'hôte à partir duquel il se connecte.

La syntaxe utilisée pour désigner un utilisateur est donc la suivante :

```
nom_utilisateur[@nom_hôte]
```

Pour le nom d'hôte, la valeur `'%'` signifie "n'importe quel hôte" ; c'est la valeur par défaut utilisée lorsque le nom d'hôte n'est pas spécifié. Le caractère `%` peut aussi être utilisé comme caractère joker dans le nom d'hôte ou l'adresse IP pour spécifier une liste de machine (`oheurtel@'%.olivier-heurtel.fr'`) ou une plage d'adresses IP (`oheurtel@'192.168.1.%'`). Le nom d'utilisateur peut être vide (utilisateur anonyme).

Il est possible d'avoir un utilisateur `nom_utilisateur@'%'` qui peut se connecter à partir de n'importe quelle machine avec certains droits et un "autre" utilisateur `nom_utilisateur@nom_hote` ayant le même nom, mais pouvant se connecter à parti d'une machine avec des droits différents (par exemple plus restrictifs si la machine est considérée comme peu sûre, ou moins restrictifs si la machine est considérée comme très sûre).

Lorsqu'un utilisateur se connecte localement (directement sur le serveur), c'est le nom d'hôte `localhost` qui est utilisé dans son identification ; ce nom d'hôte n'est pas considéré comme faisant partie de "tous les hôtes" défini par `'%'`. Si un utilisateur est autorisé à se connecter à partir de n'importe quelle machine, y compris le serveur, il faudra créer 2 comptes : `xxxx@'%'` et `xxxx@localhost` (et gérer correctement les droits sur les deux comptes).

Les informations sur les utilisateurs et leurs droits sont stockés dans la base de données `mysql` :

| **Table** | **Contenu** |
|---|---|
| `user` | Liste des utilisateurs avec leurs privilèges globaux (privilèges qui s'appliquent au serveur MySQL et à toutes les bases de données du serveur). |
| `db` et `host` | Liste des privilèges de niveau base de données attribués aux utilisateurs. |
| `tables_priv`, `columns_priv` et `procs_priv` | Liste des privilèges de niveau objet attribués aux utilisateurs. |

Pour gérer les utilisateurs et les droits, il faut des privilèges globaux précis (`CREATE USER`, `GRANT OPTION`, etc.). Par défaut, ces privilèges sont attribués au compte `root`, puisque ce dernier à tous les droits. Dans la suite, nous supposerons que la gestion des utilisateurs et des droits est effectuée à l'aide du compte `root` et nous ne préciserons pas quel droit est requis pour effectuer telle action. Pour en savoir plus sur ce sujet, reportez-vous à la documentation MySQL.

## 2. Gérer les utilisateurs

### a. Créer des utilisateurs

Avant la version 5.0.2, un nouvel utilisateur était créé implicitement par attribution d'un premier droit à l'aide de l'ordre SQL `GRANT` (voir ci-après).

Depuis la version 5.0.2, l'ordre SQL `CREATE USER` permet de créer explicitement un utilisateur.

*Syntaxe*

```
CREATE USER spécification_utilisateur [,...]

spécification_utilisateur =
nom_utilisateur[@nom_hôte] [IDENTIFIED BY [PASSWORD] 'mot_de_passe']
```

nom_utilisateur et mot_de_passe sont respectivement le nom et le mot de passe du nouveau compte. Si la clause IDENTIFIED BY est absente, le compte est créé sans mot de passe.

nom_hôte permet de spécifier le nom de la machine à partir de laquelle le nouvel utilisateur peut se connecter. Si cette clause est omise, la valeur par défaut '%' est utilisée ; dans ce cas, l'utilisateur peut se connecter à partir de n'importe quel hôte.

Si le mot clé PASSWORD est omis, le mot de passe doit être saisi en clair ; il sera automatiquement chiffré (haché) par MySQL à l'aide de la fonction PASSWORD avant d'être stocké dans la base mysql. Si le mot clé PASSWORD est présent, il faut saisir un mot de passe déjà chiffré (nombre hexadécimal de 41 chiffres).

*Exemple*

```
mysql> CREATE USER eniadm IDENTIFIED BY 'eni';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE USER oheurtel@localhost IDENTIFIED BY 'oh';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT host,user,password FROM mysql.user;
+-----------+----------+-------------------------------------------+
| host      | user     | password                                  |
+-----------+----------+-------------------------------------------+
| localhost | root     |                                           |
| linux     | root     |                                           |
| localhost |          |                                           |
| linux     |          |                                           |
| localhost | pma      |                                           |
| localhost | eniweb   | *7F3BF7031A324F9FA79930B44A098C84FA3FBB97 |
| %         | eniadm   | *EF068FAEFCC1D0D291B52D66CA1CAD5D3B1546F1 |
| localhost | oheurtel | *BE6735F99EBE04720DC10FB173FF7E76A881E16F |
+-----------+----------+-------------------------------------------+
8 rows in set (0.01 sec)

mysql> exit
Bye

[root@xampp ~]# mysql -u eniadm -p
Enter password:
ERROR 1045 (28000): Access denied for user 'eniadm'@'localhost' (using password: YES)
```

Cet exemple illustre le point expliqué précédemment : l'hôte localhost n'est pas considéré comme faisant partie de "tous les hôtes" défini par '%'. L'utilisateur ainsi créé ne peut pas se connecter à partir du serveur (par contre, il peut se connecter à partir de n'importe quelle autre machine).

Pour résoudre à ce problème, il est possible de créer un "deuxième" utilisateur.

*Exemple*

```
mysql> CREATE USER eniadm@localhost IDENTIFIED BY 'eni';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT host,user,password
    -> FROM mysql.user WHERE user = 'eniadm';
+-----------+--------+-------------------------------------------+
| host      | user   | password                                  |
+-----------+--------+-------------------------------------------+
| localhost | eniadm | *EF068FAEFCC1D0D291B52D66CA1CAD5D3B1546F1 |
| %         | eniadm | *EF068FAEFCC1D0D291B52D66CA1CAD5D3B1546F1 |
+-----------+--------+-------------------------------------------+
2 rows in set (0.00 sec)

mysql> exit
Bye

[root@xampp ~]# mysql -u eni -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

Ce "deuxième" utilisateur aurait pu être créé avec un mot de passe différent (éventuellement sans mot de passe).

### b. Supprimer des utilisateurs

Avant la version 4.1.1, un utilisateur était supprimé par un DELETE directement dans la table mysql.user, après révocation de tous ses droits à l'aide de l'ordre SQL REVOKE (voir ci-après).

Depuis la version 4.1.1, l'ordre SQL DROP USER permet de supprimer explicitement un utilisateur.

*Syntaxe*

```
DROP USER nom_utilisateur[@nom_hôte[,...]
```

Si le nom d'hôte est omis, la valeur par défaut '%' est utilisée.

Lors de la suppression d'un utilisateur, les droits de l'utilisateur sont aussi supprimés ; par contre, les objets éventuels créés par l'utilisateur ne

sont pas supprimés.

### c. Modifier le mot de passe des utilisateurs

L'ordre SQL SET PASSWORD permet de modifier le mot de passe d'un utilisateur.

*Syntaxe*

```
SET PASSWORD [FOR nom_utilisateur[@nom_hôte]] =
        PASSWORD('nouveau_mot_de_passe_en_clair')
      | 'nouveau_mot_de_passe_haché'
```

Si le nom d'hôte est omis, la valeur par défaut '%' est utilisée.

Si la clause FOR est omise, cette commande permet de modifier le mot de passe de l'utilisateur courant.

La fonction PASSWORD appliquée au nouveau mot de passe saisi en clair permet de le chiffrer (hacher plus précisément) avant son stockage dans la base de données mysql. Le mot de passe déjà haché peut être saisi directement dans la commande.

*Exemple*

```
mysql> SET PASSWORD FOR eniadm = PASSWORD('secret');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT host,user,password
    -> FROM mysql.user WHERE user = 'eniadm';
+-----------+--------+-------------------------------------------+
| host      | user   | password                                  |
+-----------+--------+-------------------------------------------+
| %         | eniadm | *14E65567ABDB5135D0CFD9A70B3032C179A49EE7 |
| localhost | eniadm | *EF068FAEFCC1D0D291B52D66CA1CAD5D3B1546F1 |
+-----------+--------+-------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT PASSWORD('secret');
+-------------------------------------------+
| PASSWORD('secret')                        |
+-------------------------------------------+
| *14E65567ABDB5135D0CFD9A70B3032C179A49EE7 |
+-------------------------------------------+
1 row in set (0.00 sec)

mysql> SET PASSWORD FOR eniadm@localhost =
    ->   '*14E65567ABDB5135D0CFD9A70B3032C179A49EE7';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT host,user,password
    -> FROM mysql.user WHERE user = 'eniadm';
+-----------+--------+-------------------------------------------+
| host      | user   | password                                  |
+-----------+--------+-------------------------------------------+
| %         | eniadm | *14E65567ABDB5135D0CFD9A70B3032C179A49EE7 |
| localhost | eniadm | *14E65567ABDB5135D0CFD9A70B3032C179A49EE7 |
+-----------+--------+-------------------------------------------+
2 rows in set (0.00 sec)
```

## 3. Gérer les droits des utilisateurs

### a. Attribuer des droits aux utilisateurs

L'ordre SQL GRANT permet d'attribuer des droits aux utilisateurs.

*Syntaxe simplifiée*

```
GRANT privilège,[,...]
ON spécification_cible
TO spécification_utilisateur [,...]

spécification_cible =
    *.*
  | [nom_base.]*
  | [TABLE | FUNCTION | PROCEDURE] [nom_base.]nom_objet

spécification_utilisateur =
nom_utilisateur[@nom_hôte] [IDENTIFIED BY [PASSWORD] 'mot_de_passe']
```

La clause *spécification_utilisateur* est la même que pour l'ordre CREATE USER. Si l'utilisateur (au sens couple utilisateur/hôte) n'existe pas, il est automatiquement créé, avec ou sans mot de passe selon que la clause IDENTIFIED BY est présente ou pas.

Les privilèges peuvent être attribués à trois niveaux :

### Global

Les privilèges attribués au niveau global s'appliquent à toutes les bases de données du serveur. Pour attribuer un privilège au niveau global, il faut utiliser la valeur `*.*` pour la clause *spécification_cible*.

### Base de données

Les privilèges attribués au niveau base de données s'appliquent à tous les objets d'une base de données. Pour attribuer un privilège au niveau base de données, il faut utiliser la valeur `[nom_base].*` pour la clause *spécification_cible*. Si `nom_base` est omis, les droits sont attribués pour la base de données courante.

### Objet

Les privilèges attribués au niveau objet s'appliquent à un objet d'une base de données : table, vue, programme stocké. Pour attribuer un privilège au niveau objet, il faut utiliser la valeur `[TABLE | FUNCTION | PROCEDURE] [nom_base.]nom_objet` pour la clause *spécification_cible*. Si `nom_base` est omis, l'objet est recherché dans la base de données courante. Le mot clé `TABLE`, `FUNCTION` ou `PROCEDURE` permet de préciser la nature de l'objet concerné ; pour attribuer un droit sur une table ou une vue, le mot clé `TABLE` peut être omis (c'est la valeur par défaut).

`privilège` est un nom de privilège. Les privilèges les plus souvent utilisés sont les suivants :

| **Nom** | **Signification** | **Niveau (1)** | | |
|---|---|---|---|---|
| | | **G** | **B** | **O** |
| ALL [PRIVILEGES] | Tous les droits applicables pour le niveau concerné. | x | x | x |
| ALTER | Autorise la modification des tables (ordre ALTER TABLE). Au niveau global ou base de données, autorise aussi la modification d'une base de données (ordre ALTER DATABASE). | x | x | x |
| ALTER ROUTINE | Autorise la modification ou la suppression des programmes stockés (ordres {ALTER \| DROP} {FUNCTION \| PROCEDURE}). | x | x | x |
| CREATE | Autorise la création des tables (ordre CREATE TABLE). Au niveau global ou base de données, autorise aussi la création d'une base de données (ordre CREATE DATABASE). | x | x | |
| CREATE ROUTINE | Autorise la création des programmes stockés (ordres CREATE {FUNCTION \| PROCEDURE}). | x | x | |
| CREATE TEMPORARY TABLES | Autorise la création des tables temporaires (ordre CREATE TEMPORARY TABLE). | x | x | |
| CREATE USER | Autorise la gestion des utilisateurs (ordres {CREATE \| DROP \| RENAME} USER), | x | | |
| CREATE VIEW | Autorise la création des vues (ordre CREATE VIEW). | x | x | |
| DELETE | Autorise la suppression de lignes dans les tables (ordre DELETE), | x | x | x |
| DROP | Autorise la suppression des tables et des vues (ordre DROP {TABLE \| VIEW}). Au niveau global ou base de données, | x | x | x |

| | | | | |
|---|---|---|---|---|
| | autorise aussi la suppression d'une base de données (ordre DRUP DATABASE). | | | |
| EXECUTE | Autorise l'exécution des programmes stockés (ordre CALL ou appel d'une fonction stockée). | x | x | x |
| FILE | Autorise les déchargements et chargements de données (ordres SELECT ... INTO OUTFILE et LOAD DATE INFILE). | x | | |
| GRANT OPTION | Autorise l'attribution et la révocation des privilèges (ordres GRANT et REVOKE). | x | x | x |
| INDEX | Autorise la création et la suppression des index (ordres {CREATE \| DROP} INDEX). | x | x | x |
| INSERT[(nom_colonne [,...])] | Autorise l'insertion de lignes dans les tables (ordre INSERT). | x | x | x |
| LOCK TABLES | Autorise le verrouillage des tables (ordre LOCK TABLES). Nécessite aussi le privilège SELECT sur les tables. | x | x | x |
| SELECT[(nom_colonne [,...])] | Autorise la sélection de lignes dans les tables (ordre SELECT). | x | x | x |
| SHOW DATABASES | Autorise l'affichage de la liste des bases de données (ordre SHOW DATABASES). | x | | |
| SHOW VIEW | Autorise l'affichage de la définition d'une vue (ordre SHOW CREATE VIEW). | x | x | x |
| TRIGGER | Autorise la création et la suppression des triggers (ordres {CREATE \| DROP} TRIGGER). | x | x | x |
| UPDATE[(nom_colonne [,...])] | Autorise la modification de lignes dans les tables (ordre UPDATE). | x | x | x |

(1) G = niveau global - B = niveau base de données - O = niveau objet

 Lorsqu'aucun privilège de niveau global n'est attribué à un utilisateur, ce dernier a le privilège par défaut USAGE.

Lorsque les privilèges SELECT, INSERT ou UPDATE sont attribués sur une table ou sur une vue (niveau objet), il est possible de restreindre le droit à une liste de colonnes ; seules les colonnes mentionnées pourront être lues, insérées ou modifiées.

 La commande SHOW PRIVILEGES affiche la liste de tous les privilèges supportés par MySQL.

*Exemples*

```
mysql> -- L'utilisateur "eniadm" est l'administrateur de
mysql> -- la base "eni" : il a tous les droits sur cette base.
mysql> GRANT ALL ON eni.* TO eniadm,eniadm@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> -- L'utilisateur "eniadm" a aussi le droit de lire
mysql> -- les données de la table "rubrique" de la base "demo".
```

```
mysql> GRANT SELECT ON demo.rubrique TO eniadm,eniadm@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> -- L'utilisateur "oheurtel" est un utilisateur de
mysql> -- la base "eni" : il a uniquement les droits de
mysql> -- lire et mettre à jour les données, et d'exécuter
mysql> -- les programmes stockés.
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,EXECUTE ON eni.*
    -> TO oheurtel@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> -- Donnons aussi à l'utilisateur "oheurtel" le droit
mysql> -- de charger et décharger des données.
mysql> GRANT FILE ON *.* TO oheurtel@localhost;
Query OK, 0 rows affected (0.00 sec)
```

Un utilisateur n'a pas besoin du privilège SHOW DATABASES pour lister les bases de données pour lesquelles il a un privilège quelconque.

Par ailleurs, lorsqu'un utilisateur liste les tables à l'aide de l'ordre SHOW TABLES, il ne voit que les tables sur lesquelles il a au moins un droit.

### b. Lister les droits d'un utilisateur

L'ordre SQL SHOW GRANTS permet de lister les droits d'un utilisateur.

*Syntaxe*

```
SHOW GRANTS [FOR nom_utilisateur[@nom_hôte]]
```

Si la clause FOR est omise, la commande liste les droits de la session courante.

*Exemple*

```
mysql> SHOW GRANTS FOR oheurtel@localhost;
+-----------------------------------------------------------------------------------------------------------+
| Grants for oheurtel@localhost                                                                              |
+-----------------------------------------------------------------------------------------------------------+
| GRANT FILE ON *.* TO 'oheurtel'@'localhost' IDENTIFIED BY PASSWORD '*BE6735F99EBE04720DC10FB173FF7E76A881E16F|
| GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE ON `eni`.* TO 'oheurtel'@'localhost'                           |
+-----------------------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

Comme vous pouvez le constater sur cet exemple, les droits sont affichés sous la forme d'une liste de commandes d'attribution de privilèges.

Pour avoir une présentation différente, il faut interroger directement les tables de la base mysql.

### c. Révoquer des droits d'un utilisateur

L'ordre SQL REVOKE permet de révoquer des droits d'un utilisateur.

*Syntaxe 1*

```
REVOKE privilège,[,...]
ON spécification_cible
FROM nom_utilisateur[@nom_hôte][,...]

spécification_cible =
    *.*
  | [nom_base.]*
  | [TABLE | FUNCTION | PROCEDURE] [nom_base.]nom_objet
```

*spécification_cible* a la même signification que pour l'ordre SQL GRANT.

*Syntaxe 2*

```
REVOKE ALL PRIVILEGES, GRANT OPTION
FROM nom_utilisateur[@nom_hôte][,...]
```

Cette deuxième syntaxe permet de révoquer facilement tous les droits d'un utilisateur, à tous les niveaux (global, base de données et objet).

MySQL ne supprime pas automatiquement les privilèges lorsqu'une table, une vue ou une base de données est supprimée. Par contre, lorsqu'un programme stocké est supprimé, les privilèges attribués sur le programme sont supprimés. Lorsqu'un utilisateur est supprimé, les privilèges qui lui étaient attribués sont supprimés.

*Exemple*

```
mysql> -- Finalement, l'utilisateur "oheurtel" n'a plus le droit
mysql> -- de charger et décharger des données.
mysql> REVOKE FILE ON *.* FROM oheurtel@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW GRANTS FOR oheurtel@localhost;
```

```
+------------------------------------------------------------------------------------------------------------+
| Grants for oheurtel@localhost                                                                              |
+------------------------------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'oheurtel'@'localhost' IDENTIFIED BY PASSWORD '*BE6735F99EBE04720DC10FB173FF7E76A881E16F|
| GRANT SELECT, INSERT, UPDATE, DELETE, EXECUTE ON `eni`.* TO 'oheurtel'@'localhost'                         |
+------------------------------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

# Gérer les tables

## 1. Créer une table

L'ordre SQL `CREATE TABLE` permet de créer une nouvelle table.

*Syntaxe simplifiée*

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table
(
spécification_colonne,
...
)
[ENGINE|TYPE[=] moteur]

spécification_colonne =
   nom_colonne type_colonne [option_colonne]

option_colonne =
   [NOT NULL | NULL] [DEFAULT valeur] [AUTO_INCREMENT]
   [[PRIMARY] KEY] [UNIQUE [KEY]]
```

*Exemple simple*

```
mysql> CREATE TABLE evenement
    ->   (
    ->   id INT,
    ->   nom VARCHAR(20)
    ->   );
Query OK, 0 rows affected (0.00 sec)

mysql> DESC evenement;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int(11)     | YES  |     | NULL    |       |
| nom   | varchar(20) | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Le nom de la nouvelle table doit respecter les règles de nommage des identifiants MySQL. Par défaut, la table est stockée dans la base de données courante ; le nom de la table peut être de la forme `nom_base.nom_table` pour la stocker dans une autre base de données.

Le mot clé `TEMPORARY` indique que la table est temporaire. Une table temporaire n'est visible que dans la connexion courante et est automatiquement supprimée lorsque la transaction courante se termine. Deux connexions différentes peuvent utiliser le même nom de table temporaire sans conflit.

La clause `IF NOT EXISTS` permet d'éviter d'obtenir une erreur si une table de même nom existe (la structure de la table n'est pas vérifiée pour savoir s'il s'agit effectivement de la même table).

La clause optionnelle `ENGINE` (anciennement `TYPE`) permet de définir le moteur de stockage utilisé pour la table. Le moteur détermine la manière dont les données sont stockées et quelles opérations sont autorisées sur les données. Les moteurs de stockage les plus utilisés sont les suivants :

`ARCHIVE`

Moteur qui stocke les données dans un format compressé. Ce moteur est intéressant pour les tables volumineuses mais la compression/décompression ralentit les opérations de lecture ou de mise à jour.

`MEMORY` (anciennement `HEAP`)

Moteur qui stocke les données en mémoire. Ce moteur est très intéressant pour les tables temporaires (mais les données sont perdues lorsque le serveur s'arrête).

`InnoDB`

Moteur permettant de gérer les clés étrangères (cf. dans ce chapitre Utiliser les clés et les index - Clé étrangère) ainsi que les transactions et le verrouillage de niveau ligne (cf. chapitre Techniques avancées avec MySQL - Gérer les transactions et les accès concurrents). Pour autoriser l'utilisation des tables InnoDB, il faut mettre en commentaire le paramètre `skip-innodb` dans le fichier de configuration de MySQL.

`MyISAM`

Moteur utilisé par défaut. Ce moteur est très performant mais il ne gère pas les transactions.

---

Chaque moteur possède ces propres paramètres dans le fichier de configuration de MySQL (voir la documentation). La commande `SHOW ENGINES` permet de voir quels sont les moteurs disponibles dans le serveur MySQL. Si un moteur est spécifié mais que celui-ci n'est pas disponible, MySQL utilise le moteur par défaut (normalement MyISAM) et génère une alerte (pas une erreur).

---

Chaque colonne de la table est définie avec au minimum un nom de colonne et un type de données. Le nom de la colonne doit respecter les règles de nommage des identifiants MySQL. Le type de données doit être un des types de données présenté au chapitre Introduction à MySQL.

En complément, chaque colonne peut avoir un ou plusieurs des attributs suivants :

`NOT NULL | NULL`

`NOT NULL` indique que la colonne est obligatoire (elle n'accepte pas les valeurs `NULL`). La valeur par défaut est `NULL` : la colonne accepte les valeurs `NULL`.

`DEFAULT valeur`

Cette clause permet de définir la valeur par défaut de la colonne ; cette valeur est insérée automatiquement dans la colonne lorsque cette dernière n'est pas présente dans un `INSERT` (ou lorsque le mot clé `DEFAULT` est utilisé comme valeur lors d'un `INSERT` ou d'un `UPDATE`). La valeur doit être une constante ; utiliser une fonction est interdit. Cette clause n'est pas autorisée pour une colonne de type `BLOB` ou `TEXT`. Si la clause `DEFAULT` est omise, le comportement de MySQL dépend du mode SQL actif (voir ci-après).

`AUTO_INCREMENT`

Cette clause indique que MySQL doit insérer une valeur entière unique dans la colonne lors de l'insertion, si aucune valeur autre que `NULL` ou `0` n'est affectée à la colonne. La numérotation commence à 1 et la valeur automatiquement insérée est égale à la valeur la plus grande actuellement stockée dans la colonne plus un. Les restrictions suivantes s'appliquent lorsqu'une colonne est de type `AUTO_INCREMENT` :

- Uniquement pour les colonnes de type entier
- Une seule colonne de ce type par table
- Clause `DEFAULT` interdite
- La colonne doit être indexée (ce qui est généralement le cas car les colonnes de ce type sont souvent définies comme clé primaire)

---

La valeur automatiquement insérée dans une colonne `AUTO_INCREMENT` peut être récupérée grâce à la fonction `LAST_INSERT_ID` (cf. chapitre Utiliser les fonctions MySQL - Fonctions système).

---

`[PRIMARY] KEY`

Cette clause indique que la colonne est la clé primaire de la table (cf. Utiliser les clés et les index - Clé primaire ou unique de ce chapitre pour plus d'informations).

`UNIQUE [KEY]`

Cette clause indique que la colonne est une clé unique (cf. Utiliser les clés et les index - Clé primaire ou unique de ce chapitre pour plus d'informations).

Si aucune valeur par défaut n'est définie explicitement pour une colonne et que la colonne autorise les valeurs `NULL`, la

valeur NULL est définie comme valeur par défaut. Par contre, si la colonne est obligatoire, le fonctionnement dépend du mode SQL actif :

- Si le mode SQL strict n'est pas actif (cas par défaut), MySQL utilise sa propre valeur par défaut liée au type de données : 0 pour un nombre, chaîne vide pour une chaîne, date "zéro" pour une date.

- Si le mode SQL strict est actif, une erreur se produit.

---

Si aucune valeur par défaut n'est spécifiée pour la première colonne de type TIMESTAMP d'une table, MySQL en définit une automatiquement égale à CURRENT_TIMESTAMP (fonction qui donne la date/heure actuelle - cf. chapitre Utiliser les fonctions MySQL - Fonctions dates).

---

*Exemple*

```
mysql> CREATE TABLE utilisateur
    ->   (
    ->   id INT PRIMARY KEY AUTO_INCREMENT,
    ->   nom VARCHAR(40) NOT NULL,
    ->   prenom VARCHAR(40) NOT NULL,
    ->   mail VARCHAR(200) NOT NULL UNIQUE KEY,
    ->   mot_de_passe BLOB NOT NULL,
    ->   est_actif BOOLEAN NOT NULL DEFAULT TRUE,
    ->   date_maj TIMESTAMP
    ->   );
Query OK, 0 rows affected (0.02 sec)

mysql> DESC utilisateur;
+--------------+--------------+------+-----+-------------------+----------------+
| Field        | Type         | Null | Key | Default           | Extra          |
+--------------+--------------+------+-----+-------------------+----------------+
| id           | int(11)      | NO   | PRI | NULL              | auto_increment |
| nom          | varchar(40)  | NO   |     |                   |                |
| prenom       | varchar(40)  | NO   |     |                   |                |
| mail         | varchar(200) | NO   | UNI |                   |                |
| mot_de_passe | blob         | NO   |     |                   |                |
| est_actif    | tinyint(1)   | NO   |     | 1                 |                |
| date_maj     | timestamp    | NO   |     | CURRENT_TIMESTAMP |                |
+--------------+--------------+------+-----+-------------------+----------------+
7 rows in set (0.01 sec)
```

## 2. Créer une table par copie

L'ordre SQL CREATE TABLE possède deux variantes qui permettent de créer une table, soit par copie d'une autre table, soit par copie du résultat d'une requête SELECT.

### Copie d'une autre table

*Syntaxe*

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table
{ LIKE nom_autre_table | (LIKE nom_autre_table) }
```

Cette commande crée une table vide dont la définition est identique à une autre table (y compris les attributs des colonnes et les index définis sur la table d'origine).

*Exemple*

```
mysql> CREATE TABLE responsable_collection LIKE auteur;
Query OK, 0 rows affected (0.00 sec)

mysql> DESC responsable_collection;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
```

```
| id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom          | varchar(40)  | NO   | MUL |         |                |
| prenom       | varchar(40)  | NO   |     |         |                |
| mail         | varchar(200) | YES  |     | NULL    |                |
| tel_bureau   | varchar(10)  | YES  |     | NULL    |                |
| tel_portable | varchar(10)  | YES  |     | NULL    |                |
| tel_domicile | varchar(10)  | YES  |     | NULL    |                |
| mot_de_passe | blob         | YES  |     | NULL    |                |
| profil       | blob         | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

## Copie du résultat d'une requête SELECT

*Syntaxe*

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table
[(spécification_colonne[,...])]
[ENGINE|TYPE[=] moteur]
[IGNORE | REPLACE]
[AS] requête_SELECT
```

La clause optionnelle *spécification_colonne* permet de définir explicitement des colonnes de la nouvelle table. Dans cette clause, les colonnes sont définies avec la même syntaxe que dans l'ordre CREATE TABLE présenté dans la section précédente.

Si la clause *spécification_colonnes* est omise, la table est créée avec la structure correspondant au résultat de la requête SELECT et le résultat de la requête SELECT est inséré dans la nouvelle table.

Si la clause *spécification_colonnes* est présente, la structure correspondant au résultat de la requête SELECT est ajoutée à la définition de la table et le résultat de la requête SELECT est inséré dans la nouvelle table, avec les valeurs par défaut affectées aux colonnes explicitement définies. La liste des colonnes explicitement définies peut comporter une colonne de même nom qu'une colonne de la requête SELECT ; dans ce cas, cette colonne est alimentée par la colonne homonyme de la requête SELECT (la colonne conserve le type de données qui lui a été explicitement attribué).

 Pensez à utiliser des alias de colonnes pour les expressions de la requête SELECT.

Dans les deux cas, la nouvelle table est créée sans clé (primaire ou unique) et sans index.

Si la clause IF NOT EXISTS est présente et qu'une table de même nom existe déjà, le résultat de la requête SELECT est quand même inséré dans la table. Dans ce cas, si une ligne insérée provoque un doublon dans une clé primaire ou unique, une erreur est retournée, sauf si la clause optionnelle IGNORE ou REPLACE est utilisée : IGNORE permet de rejeter les lignes qui provoque un doublon et REPLACE permet de remplacer l'ancienne ligne par la nouvelle.

*Exemple*

```
mysql> CREATE TABLE categorie(id INT PRIMARY KEY AUTO_INCREMENT)
    -> AS SELECT titre nom FROM rubrique WHERE id_parent IS NULL;
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> DESC categorie;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int(11)     | NO   | PRI | NULL    | auto_increment |
| nom   | varchar(20) | NO   |     |         |                |
+-------+-------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM categorie;
+----+-----------------+
| id | nom             |
+----+-----------------+
|  1 | Base de données |
|  2 | Développement   |
|  3 | Internet        |
|  4 | Open Source     |
```

```
|  5 | Certification   |
+----+-----------------+
5 rows in set (0.00 sec)
```

## 3. Renommer une table

L’ordre SQL RENAME TABLE permet de renommer une table.

*Syntaxe*

```
RENAME TABLE ancien_nom TO nouveau_nom
```

*Exemple*

```
mysql> RENAME TABLE utilisateur TO client;
Query OK, 0 rows affected (0.00 sec)
```

Avec cette syntaxe, il est possible de déplacer une table d’une base à une autre, sous réserve d’avoir les droits suffisants.

Une variante de cette syntaxe permet de renommer plusieurs tables en une seule commande (un ancien nom pouvant même tout de suite être réutilisé).

*Syntaxe*

```
RENAME TABLE a TO temp, b TO a, temp TO b;
```

Une table peut aussi être renommée par un ordre ALTER TABLE.

*Syntaxe*

```
ALTER TABLE ancien_nom RENAME [AS | TO] nouveau_nom
```

## 4. Modifier la structure d’une table

L’ordre SQL ALTER TABLE peut être utilisé pour modifier la structure d’une table. Cet ordre permet notamment :

- D’ajouter des colonnes ;
- De supprimer des colonnes ;
- De modifier les attributs des colonnes.

La commande ALTER TABLE offre d’autres possibilités (modifier le moteur d’une table notamment). Pour en savoir plus, consultez la documentation MySQL.

### Ajouter une ou plusieurs colonnes

*Syntaxe*

```
ALTER TABLE nom_table ADD [COLUMN] spécification_colonne
[FIRST | AFTER nom_colonne]

ALTER TABLE nom_table ADD [COLUMN] (spécification_colonne[,...])
```

La clause *spécification_colonne* permet de définir les caractéristiques des colonnes ajoutées dans la table. La syntaxe est la même que dans l’ordre CREATE TABLE.

La première syntaxe permet d’ajouter une seule colonne, éventuellement à un endroit précis spécifié par la clause FIRST ou AFTER ; par défaut, la colonne est ajoutée à la fin de la table.

La deuxième syntaxe permet d’ajouter plusieurs colonnes, à la fin de la table.

*Exemple*

```
mysql> ALTER TABLE evenement ADD
    ->   (
    ->   date_debut DATE,
    ->   date_fin DATE,
    ->   id_collection INT DEFAULT 1,
    ->   pays VARCHAR(50)
    ->   );
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| id            | int(11)     | YES  |     | NULL    |       |
| nom           | varchar(20) | YES  |     | NULL    |       |
| date_debut    | date        | YES  |     | NULL    |       |
| date_fin      | date        | YES  |     | NULL    |       |
| id_collection | int(11)     | YES  |     | 1       |       |
| pays          | varchar(50) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)
```

Si une colonne `AUTO_INCREMENT` est ajoutée à une table qui contient des données, la colonne est automatiquement alimentée par des nombres à partir de `1`.

Si une colonne ayant une clause `DEFAULT` est ajoutée à une table qui contient des données, la colonne est automatiquement alimentée avec la valeur par défaut.

Si une colonne `NOT NULL` est ajoutée à une table qui contient des données, la colonne est automatiquement alimentée avec la valeur par défaut de la colonne (valeur par défaut implicite si aucune clause `DEFAULT` explicite n'est définie).

### Supprimer une colonne

*Syntaxe*

```
ALTER TABLE nom_table DROP [COLUMN] nom_colonne
```

*Exemple*

```
mysql> ALTER TABLE evenement DROP id_collection;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| nom        | varchar(20) | YES  |     | NULL    |       |
| date_debut | date        | YES  |     | NULL    |       |
| date_fin   | date        | YES  |     | NULL    |       |
| pays       | varchar(50) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Si la colonne fait partie d'un index, elle est aussi supprimée de l'index ; si c'était la seule colonne de l'index, l'index est supprimé.

### Modifier ou supprimer la valeur par défaut d'une colonne

*Syntaxe*

```
ALTER TABLE nom_table ALTER [COLUMN] nom_column
   {SET DEFAULT valeur | DROP DEFAULT}
```

*Exemple*

```
mysql> ALTER TABLE evenement ALTER pays SET DEFAULT 'FRANCE';
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| nom        | varchar(20) | YES  |     | NULL    |       |
| date_debut | date        | YES  |     | NULL    |       |
| date_fin   | date        | YES  |     | NULL    |       |
| pays       | varchar(50) | YES  |     | FRANCE  |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

## Modifier les attributs des colonnes

*Syntaxe*

```
ALTER TABLE nom_table CHANGE [COLUMN]
ancien_nom_colonne nouveau_nom_colonne type_colonne [option_colonne]

ALTER TABLE nom_table MODIFY [COLUMN]
nom_colonne type_colonne [option_colonne]
```

La première syntaxe permet de modifier les attributs d'une colonne, y compris son nom. La deuxième syntaxe permet de modifier les attributs d'une colonne en conservant le même nom de colonne.

La clause *option_colonne* permet de définir les nouvelles options de la colonne (`NOT NULL`, `DEFAULT`, etc). La syntaxe est la même que dans l'ordre `CREATE TABLE`. Si vous ne souhaitez pas modifier une option de la colonne, vous devez la remettre à l'identique dans l'ordre `ALTER TABLE`, sinon l'option en question est supprimée.

*Exemple*

```
mysql> ALTER TABLE evenement CHANGE
    ->   pays code_pays VARCHAR(50) DEFAULT 'FR';
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| nom        | varchar(20) | YES  |     | NULL    |       |
| date_debut | date        | YES  |     | NULL    |       |
| date_fin   | date        | YES  |     | NULL    |       |
| code_pays  | varchar(50) | YES  |     | FR      |       |
+------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE evenement MODIFY
    ->   code_pays CHAR(2) NOT NULL;
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| id         | int(11)     | YES  |     | NULL    |       |
| nom        | varchar(20) | YES  |     | NULL    |       |
| date_debut | date        | YES  |     | NULL    |       |
| date_fin   | date        | YES  |     | NULL    |       |
| code_pays  | char(2)     | NO   |     |         |       |
```

```
+------------+------------+------+-----+--------+-------+
5 rows in set (0.00 sec)

mysql> ALTER TABLE evenement MODIFY id INT AUTO_INCREMENT;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto
column and it must be defined as a key
```

Le dernier exemple montre qu'une colonne ne peut pas avoir l'attribut AUTO_INCREMENT si elle n'est pas indexée.

Si une colonne devient obligatoire (NOT NULL), les valeurs NULL éventuellement présentes dans la colonne sont remplacées par la valeur par défaut implicite associée au type.

---

En cas de modification du type de données, MySQL tente de convertir les données au mieux vers le nouveau type. Si la longueur d'une colonne de type chaîne est diminuée, les valeurs actuellement stockées dans la colonne sont éventuellement tronquées pour s'adapter à la nouvelle longueur. De même, si la longueur d'une colonne numérique est diminuée, les valeurs trop grandes actuellement stockées dans la colonne sont éventuellement remplacées par la valeur maximum permise par la nouvelle longueur.

---

## 5. Supprimer une table

L'ordre SQL DROP TABLE permet de supprimer une table (ou plusieurs tables).

*Syntaxe*

```
DROP [TEMPORARY] TABLE [IF EXISTS] nom_table[,...]
```

Lorsqu'une table est supprimée, les privilèges attribués aux utilisateurs sur cette table ne sont pas supprimés.

La clause IF EXISTS permet d'éviter d'obtenir une erreur si une table à supprimer n'existe pas.

# Utiliser les clés et les index

## 1. Clé primaire ou unique

### a. Définition

Une clé primaire (ou contrainte de clé primaire) garantit qu'il n'y aura jamais deux lignes dans la table qui auront la même valeur dans la(les) colonne(s) qui compose(nt) la clé. Par ailleurs, toutes les colonnes de la clé primaire sont obligatoires (clause NOT NULL implicite pour les colonnes concernées). Une seule clé primaire est autorisée par table.

Une clé unique (ou contrainte de clé unique) garantit qu'il n'y aura jamais deux lignes dans la table qui auront la même valeur dans la(les) colonne(s) qui compose(nt) la clé. À la différence de la clé primaire, les colonnes qui composent la clé unique ne sont pas forcément obligatoires ; pour les colonnes de la clé qui ne sont pas obligatoires, plusieurs lignes peuvent avoir la valeur NULL sans violer la contrainte. Plusieurs clés uniques sont autorisées par table.

Une clé primaire ou unique peut être constituée d'une seule colonne ou de plusieurs colonnes.

Lors d'une insertion ou d'une modification, une erreur se produit si une clé (primaire ou unique) contient une valeur qui existe déjà dans la table.

---

Les clés primaires et uniques sont des index particuliers qui imposent une contrainte d'unicité. Un accès par une clé primaire ou unique est donc performant.

---

### b. Gestion

Une clé primaire ou unique constituée d'une seule colonne peut être définie directement dans les options de la colonne, lors du CREATE TABLE, ou lors d'un ALTER TABLE ... {CHANGE|MODIFY} (voir la syntaxe de ces différents ordres dans les sections précédentes).

Sinon, d'une manière plus générale, une clé primaire ou unique, mono colonne ou multi colonne, peut être déclarée par une clause spécifique de définition de contrainte.

*Syntaxe*

```
[CONSTRAINT [nom_contrainte]] PRIMARY KEY (nom_colonne[,...])

[CONSTRAINT [nom_contrainte]] UNIQUE [INDEX|KEY] (nom_colonne[,...])
```

nom_contrainte est le nom attribué à la contrainte. Pour une clé primaire, ce nom est ignoré ; une clé primaire s'appelle obligatoirement PRIMARY. Pour une clé unique, le nom par défaut de la contrainte est égal au nom de la première colonne de la clé, avec si besoin un suffixe (_2, etc.) pour l'unicité.

La clause précédente peut être utilisée dans un ordre CREATE TABLE, après la liste des colonnes, ou dans un ordre ALTER TABLE ... ADD.

*Exemple*

```
<$I[]CREATE TABLE;PRIMARY KEY>mysql> CREATE TABLE collection_evenement
    ->    (
    ->    id_evenement INT,
    ->    id_collection INT,
    ->    PRIMARY KEY (id_evenement,id_collection)
    ->    );
Query OK, 0 rows affected (0.00 sec)

mysql> DESC collection_evenement;
+---------------+---------+------+-----+---------+-------+
| Field         | Type    | Null | Key | Default | Extra |
+---------------+---------+------+-----+---------+-------+
| id_evenement  | int(11) | NO   | PRI | 0       |       |
| id_collection | int(11) | NO   | PRI | 0       |       |
+---------------+---------+------+-----+---------+-------+
2 rows in set (0.01 sec)mysql> ALTER TABLE evenement ADD
```

```
    ->   (
    ->   CONSTRAINT pk_evenement PRIMARY KEY(id),
    ->   CONSTRAINT uk_nom UNIQUE KEY(nom)
    ->   );
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> -- L'attribut AUTO_INCREMENT peut maintenant être
mysql> -- affecté la colonne "id".
mysql> ALTER TABLE evenement MODIFY id INT AUTO_INCREMENT;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC evenement;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| nom        | varchar(20) | YES  | UNI | NULL    |                |
| date_debut | date        | YES  |     | NULL    |                |
| date_fin   | date        | YES  |     | NULL    |                |
| code_pays  | char(2)     | NO   |     |         |                |
+------------+-------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)mysql> ALTER TABLE auteur ADD
    ->   UNIQUE KEY(nom,prenom);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> DESC auteur;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom          | varchar(40)  | NO   | MUL |         |                |
| prenom       | varchar(40)  | NO   |     |         |                |
| mail         | varchar(200) | YES  |     | NULL    |                |
| tel_bureau   | varchar(10)  | YES  |     | NULL    |                |
| tel_portable | varchar(10)  | YES  |     | NULL    |                |
| tel_domicile | varchar(10)  | YES  |     | NULL    |                |
| mot_de_passe | blob         | YES  |     | NULL    |                |
| profil       | blob         | YES  |     | NULL    |                |
+--------------+--------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

---

 Dans une clé primaire ou unique définie sur une colonne de type chaîne, les espaces de fin de chaîne sont ignorés. Ainsi, la chaîne 'abc ' est considérée comme égale à la chaîne 'abc'.'

---

L'ordre SQL `SHOW INDEX` permet de lister les clés (et plus généralement les index) d'une table.

*Syntaxe*

```
SHOW INDEX FROM nom_table [FROM nom_base]
```

*Exemple*

```
mysql> SHOW INDEXES FROM evenement;
+-----------+------------+----------+--------------+-------------
+-----------+-------------+----------+--------+------+------------
+---------+
| Table     | Non_unique | Key_name | Seq_in_index | Column_name
| Collation | Cardinality | Sub_part | Packed | Null | Index_type
| Comment |
+-----------+------------+----------+--------------+-------------
+-----------+-------------+----------+--------+------+------------
+---------+
| evenement |          0 | PRIMARY  |            1 | id
| A         |           0 |     NULL | NULL   |      | BTREE
```

```
|          |
| evenement |           0 | uk_nom    |             1 | nom
| A         |        NULL |      NULL | NULL     | YES   | BTREE
|          |
+-----------+------------+----------+-------------+-------------
+-----------+------------+----------+--------+------+------------
+---------+

2 rows in set (0.00 sec)
```

Une clé primaire ou unique peut être supprimée par un ordre ALTER TABLE ... DROP.

*Syntaxe*

```
ALTER TABLE nom_table DROP PRIMARY KEY

ALTER TABLE nom_table DROP KEY nom_contrainte
```

*Exemple*

```
mysql> ALTER TABLE auteur DROP KEY nom;
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

## 2. Index

### a. Définition

Un index est une structure définie sur une clé constituée d'une ou plusieurs colonnes d'une table, et qui permet un accès rapide aux lignes de la table lors d'une recherche basée sur la clé de l'index. Cette structure est automatiquement maintenue et utilisée par MySQL.

Lorsque la clé de l'index est composée de plusieurs colonnes, on parle d'index composé ou concaténé.

 Les clés primaires et uniques sont des index particuliers qui imposent une contrainte d'unicité. Un accès par une clé primaire ou unique est donc performant.

### b. Gestion

Les index peuvent être créés lors de la création initiale de la table (dans l'ordre CREATE TABLE) ou ultérieurement (par un ordre ALTER TABLE ou un ordre CREATE INDEX).

La clause INDEX permet de créer un index dans un ordre CREATE TABLE ou ALTER TABLE.

*Syntaxe*

```
INDEX [nom_index] (nom_colonne[,...])
```

nom_index est le nom attribué à l'index. Le nom par défaut de l'index est égal au nom de la première colonne de la clé, avec si besoin un suffixe (_2, etc.) pour l'unicité.

La clause précédente peut être utilisée dans un ordre CREATE TABLE, après la liste des colonnes, ou dans un ordre ALTER TABLE ... ADD.

*Exemple*

```
mysql>  ALTER TABLE rubrique ADD INDEX (id_parent);
Query OK, 15 rows affected (0.01 sec)
Records: 15  Duplicates: 0  Warnings: 0

mysql> DESC rubrique;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
```

```
| id        | int(11)     | NO   | PRI | NULL    | auto_increment |
| titre     | varchar(20) | NO   |     |         |                |
| id_parent | int(11)     | YES  | MUL | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

L'ordre CREATE INDEX permet aussi de créer un index sur une table existante.

*Syntaxe*

```
CREATE [UNIQUE] INDEX nom_index ON nom_table(nom_cololonne[,...])
```

nom_index est le nom attribué à l'index.

Le mot clé optionnel UNIQUE permet de créer un index unique ; un index unique est fonctionnellement équivalent à une clé unique.

Un index peut être créé sur les premiers caractères d'une chaîne, ce qui permet d'économiser de l'espace de stockage. Pour faire cela, dans la syntaxe de définition de l'index, il suffit de mentionner la longueur souhaitée entre parenthèses dernière le nom de la colonne.

*Exemple*

```
mysql> CREATE INDEX ix_annee ON livre(annee_parution);
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX ix_nom_prenom ON auteur(nom,prenom);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> CREATE INDEX ix_mot_de_passe ON auteur(mot_de_passe(8));
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Il n'est pas possible de créer un index sur la totalité d'une colonne de type TEXT ou BLOB. Par contre, sur de telles colonnes, il est possible de créer un index sur les premiers caractères à l'aide de la syntaxe précédente.

Un index peut être supprimé par un ordre ALTER TABLE ... DROP ou un ordre DROP INDEX.

*Syntaxe*

```
ALTER TABLE nom_table DROP INDEX nom_index

DROP INDEX nom_index ON nom_table
```

*Exemple*

```
mysql> DROP INDEX ix_mot_de_passe ON auteur;
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

L'ordre SHOW INDEX présenté dans la section précédente permet de lister les index d'une table.

### c. Considérations

Les index sont automatiquement utilisés par MySQL dans plusieurs situations :

- pour trouver rapidement les lignes qui correspondent à une clause WHERE ;
- pour trouver les lignes d'une autre table dans une jointure ;
- pour trouver les valeurs MIN() et MAX() d'une colonne indexée ;

- pour trier ou grouper les lignes selon une colonne indexée.

Les index composés sont utilisés si la colonne de tête de la clé d'index est présente dans la clause `WHERE`.

En supposant qu'un index existe sur les colonnes (`nom,prenom`), les clauses suivantes utilisent l'index :

```
WHERE nom = 'HEURTEL' AND prenom = 'Olivier'
WHERE prenom = 'Olivier' AND nom = 'HEURTEL'
WHERE nom = 'HEURTEL'
```

Par contre, l'index précédent n'est pas utilisé pour la recherche suivante :

```
WHERE prenom = 'Olivier'
```

Dans certains cas, si toutes les colonnes utilisées dans une requête sont présentes dans la clé d'un index, MySQL n'a même pas besoin d'accéder à la table pour traiter la requête. En supposant qu'un index existe sur les colonnes (`nom,prenom`), MySQL n'a pas besoin d'accéder à la table pour traiter la requête suivante :

```
SELECT prenom FROM auteur WHERE nom = 'HEURTEL'
```

---

Utiliser un index composé améliore la sélectivité de l'index. Avec un index composé créé sur 2 colonnes, MySQL peut extraire directement les lignes avec l'index lorsque la recherche porte sur les deux colonnes. Si deux index séparés sont créés sur chaque colonne, MySQL va utiliser l'index le plus restrictif pour extraire les lignes qui répondent au critère correspondant avant d'éliminer ensuite les lignes qui ne répondent pas au deuxième critère.

---

Dans une clause `WHERE`, appliquer une fonction à une colonne indexée, ou utiliser une colonne indexée dans une expression, empêche MySQL d'utiliser l'index. Il en est de même si MySQL effectue une conversion implicite de la colonne indexée vers un autre type de données.

*Exemple*

```
SELECT nom,prenom,mail FROM auteur WHERE LEFT(nom,4) = 'HEUR';
```

Dans une recherche avec l'opérateur `LIKE`, MySQL ne peut utiliser un index que s'il a le début de la chaîne. Ainsi, une clause du type `LIKE 'HEUR%'` peut utiliser un index, mais pas une clause du type `LIKE '%TEL'`.

Si une requête `SELECT` retourne un grand pourcentage des lignes de la table, MySQL peut choisir de ne pas utiliser d'index s'il estime qu'un parcours complet de la table est plus performant.

---

L'ordre SQL `EXPLAIN` permet d'afficher le plan d'exécution d'une requête et de savoir si MySQL utilise ou non les index. Reportez-vous à la documentation pour en savoir plus sur le sujet.

---

Les index présentent deux inconvénients :

- ils nécessitent un espace de stockage supplémentaire ;
- ils ralentissent les opérations de mise à jour car chaque index doit lui aussi être mis à jour.

## 3. Clé étrangère

### a. Définition

Une contrainte de clé étrangère permet de garantir que chaque valeur stockée dans une clé étrangère d'une table "enfant" existe bien dans une clé (généralement primaire ou unique) précédemment définie de la table "parent".

Les contraintes de clé étrangère permettent aussi de contrôler ce qui se passe lorsqu'une ligne de la table parent est supprimée ou que la clé primaire d'une ligne de la table est modifiée.

Les contraintes de clé étrangère sont actuellement implémentées uniquement dans le moteur InnoDB ; la table référencée et la table enfant doivent être des tables InnoDB.

### b. Gestion

La clause CONSTRAINT permet de définir une contrainte de clé étrangère.

*Syntaxe*

```
[CONSTRAINT [nom_contrainte]] FOREIGN KEY (nom_colonne[,...])
REFERENCES nom_table_parent (nom_colonne[,...])
[ON DELETE {RESTRICT | NO ACTION | CASCADE | SET NULL }]
[ON UPDATE {RESTRICT | NO ACTION | CASCADE | SET NULL }]
```

nom_contrainte est le nom attribué à la contrainte ; il doit être unique dans toute la base de données. Si aucun nom n'est défini, MySQL attribue un nom par défaut à la contrainte.

La clause FOREIGN KEY permet de définir le nom des colonnes qui constituent la clé étrangère. La clause REFERENCES permet de spécifier la clé référencée par la contrainte.

Les colonnes correspondantes de la clé étrangère et de la clé référencée doivent être du même type ; les deux clés doivent être indexées, avec les colonnes dans le même ordre (l'index peut éventuellement contenir d'autres colonnes à droite). L'index sur la clé étrangère est créé automatiquement s'il n'existe pas déjà.

La clé référencée n'est pas obligatoirement primaire ou unique ; c'est une déviation de MySQL par rapport au standard SQL. Cependant, la plupart du temps, une clé étrangère référence une clé primaire (ou éventuellement une clé unique).

Les clauses ON DELETE et ON UPDATE permettent de définir ce qui se passe si une clé référencée est supprimée ou modifiée et qu'il existe une ou plusieurs lignes qui correspondent dans la table enfant. MySQL propose 4 options :

RESTRICT, NO ACTION

La modification ou la suppression est refusée s'il existe des lignes qui correspondent dans la table enfant. C'est l'option par défaut.

CASCADE

Les lignes qui correspondent dans la table enfant sont modifiées ou supprimées.

SET NULL

La clé étrangère est mise à NULL (possible uniquement si les colonnes correspondantes autorisent les NULL).

La clause CONSTRAINT peut être utilisée dans un ordre CREATE TABLE, après la liste des colonnes, ou dans un ordre ALTER TABLE ... ADD.

Un ordre ALTER TABLE peut être utilisé pour supprimer une contrainte de clé étrangère.

*Syntaxe*

```
ALTER TABLE nom_table DROP FOREIGN KEY nom_contrainte
```

*Exemple complet*

```
mysql> -- Création des tables "commande" et "ligne_commande".
mysql> CREATE TABLE commande
    ->   (
    ->   id INT PRIMARY KEY AUTO_INCREMENT,
    ->   date_commande DATE
    ->   )
    ->   ENGINE innodb;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ligne_commande
    ->   (
    ->   id_commande INT,
    ->   numero_ligne INT,
    ->   article VARCHAR(50),
    ->   quantite INT,
    ->   prix_unitaire DECIMAL(8,2),
    ->   PRIMARY KEY(id_commande,numero_ligne)
    ->   )
```

```
    ->   ENGINE innodb;
Query OK, 0 rows affected (0.01 sec)

mysql> -- Création d'une contrainte de clé étrangère entre
mysql> -- "ligne_commande" et "commande".
mysql> ALTER TABLE ligne_commande ADD
    ->   FOREIGN KEY (id_commande)
    ->   REFERENCES commande(id);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> -- Insertion d'une commande avec une ligne de commande.
mysql> INSERT INTO commande(date_commande) VALUES(NOW());
Query OK, 1 row affected (0.01 sec)

mysql> SET @id = LAST_INSERT_ID();
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,1,'PHP 5.2',1,25);
Query OK, 1 row affected (0.01 sec)

mysql> -- Insertion d'une ligne de commande avec un identifiant
mysql> -- de commande qui n'existe pas => erreur.
mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (2,1,'MySQL 5',1,25);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`eni/ligne_commande`, CONSTRAINT `ligne_commande_ibfk_1`
FOREIGN KEY (`id_commande`) REFERENCES `commande` (`id`))

mysql> -- Suppression de la commande => erreur (par défaut,
mysql> -- la suppression d'un parent est interdite s'il a des
mysql> -- enfants).
mysql> DELETE FROM commande WHERE id = @id;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
constraint fails (`eni/ligne_commande`, CONSTRAINT `ligne_commande_ibfk_1`
FOREIGN KEY (`id_commande`) REFERENCES `commande` (`id`))

mysql> -- Recréation de la contrainte de clé étrangère pour avoir
mysql> -- une suppression en cascade.
mysql> ALTER TABLE ligne_commande
    ->   DROP FOREIGN KEY ligne_commande_ibfk_1;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE ligne_commande ADD
    ->   FOREIGN KEY (id_commande)
    ->   REFERENCES commande(id)
    ->   ON DELETE CASCADE;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> -- Suppression de la commande => OK.
mysql> DELETE FROM commande WHERE id = @id;
Query OK, 1 row affected (0.06 sec)

mysql> SELECT COUNT(*) FROM ligne_commande WHERE id_commande = @id;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
```

# Utiliser des vues

## 1. Définition

Une vue est une requête SELECT dont la définition est stockée avec un nom dans la base de données.

La vue ne stocke pas de données ; les données présentées par la vue sont dérivées des tables interrogées par la requête de la vue.

Une vue s'utilise comme une table. Certaines vues peuvent être utilisées dans des ordres de mise à jour (INSERT, UPDATE, DELETE) pour modifier les données des tables sous-jacentes. Pour qu'une vue puisse être utilisée en mise à jour, il faut qu'il y ait une relation un-un entre les lignes retournées par la vue et les lignes de la table sous-jacente. Une vue ne peut pas être utilisée en mise à jour si la requête qui la définit contient notamment une des constructions suivantes :

- Une clause DISTINCT ;
- Des expressions dans la clause SELECT (interdit les ordres INSERT mais pas les ordres UPDATE qui modifient uniquement les colonnes qui ne sont pas calculées) ;
- Des agrégats, des unions, des sous-requêtes dans la clause SELECT (cf. chapitre Techniques avancées avec MySQL - Utiliser des sous-requêtes).

Les vues sont utilisées principalement pour :

- Faciliter l'accès aux données
  - La requête utilisée pour définir la vue peut être complexe et comporter par exemple des jointures entre plusieurs tables. Les requêtes qui utilisent la vue sont alors plus simples à écrire.
- Sécuriser l'accès aux données
  - La requête utilisée pour définir la vue peut masquer des colonnes (colonnes non sélectionnées dans la clause SELECT) et/ou des lignes (clause WHERE qui filtre les données). En terme de gestion de droit, il est possible de donner un droit de lecture sur une vue à un utilisateur sans lui donner le droit de lecture sur les tables sous-jacentes. Ainsi, un tel utilisateur ne pourra jamais voir ce qu'il n'a pas le droit de voir.

## 2. Gestion

L'ordre SQL CREATE VIEW permet de créer une vue.

*Syntaxe simplifiée*

```
CREATE [OR REPLACE] VIEW nom_vue [(nom_colonne[,...])]
AS requête_SELECT
[WITH CHECK OPTION]
```

nom_vue est le nom de la vue. Ce nom doit respecter les règles de nommage des identifiants MySQL. Les tables et les vues partagent le même espace de nommage à l'intérieur d'une base de données ; en conséquence, une base de données ne peut pas contenir une table et une vue qui portent le même nom.

Par défaut, les noms (ou alias) des colonnes sélectionnées dans la requête SELECT de la vue sont utilisés comme noms des colonnes de la vue. Pour nommer différemment les colonnes de la vue, il est possible de spécifier une liste de noms de colonnes derrière le nom de la vue ; cette liste doit comporter un nom pour chaque colonne/expression retournée par la requête SELECT.

La clause optionnelle OR REPLACE permet de recréer la vue si elle existe déjà.

Toutes les clauses habituelles peuvent être manipulées dans la requête SELECT utilisée pour la définition de la vue (WHERE, ORDER BY, jointures, etc.) ; une vue peut interroger d'autres vues, mais elle ne peut pas contenir de sous-requête dans la clause FROM (cf. chapitre Techniques avancées avec MySQL - Utiliser des sous-requêtes). La requête

peut interroger des tables ou des vues d'une autre base de données avec la notation habituelle (préfixer par le nom de la base de données).

*Exemple*

```
mysql> CREATE OR REPLACE VIEW
    ->   liste_rubriques(rubrique,sous_rubrique)
    -> AS
    -> SELECT par.titre rubrique,enf.titre sous_rubrique
    -> FROM rubrique par,rubrique enf
    -> WHERE enf.id_parent = par.id;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM liste_rubriques;
+-----------------+-----------------+
| rubrique        | sous_rubrique   |
+-----------------+-----------------+
| Base de données | MySQL           |
| Base de données | Oracle          |
| Développement   | Langages        |
| Développement   | Méthode         |
| Internet        | HTML - XML      |
| Internet        | Conception Web  |
| Internet        | Sécurité        |
| Open Source     | Système         |
| Open Source     | Langages        |
| Open Source     | Base de données |
+-----------------+-----------------+
10 rows in set (0.00 sec)
```

Par défaut, lorsqu'une vue est utilisée en mise à jour, il est possible d'insérer ou modifier des lignes à travers la vue qui ne respectent pas la clause WHERE de définition de la vue ; les lignes ainsi insérées ou modifiées ne pourront pas être interrogées par la vue ! Pour garantir que les lignes insérées ou modifiées à travers une vue respectent la clause WHERE de la vue et seront donc interrogeables par la vue, il faut ajouter la clause optionnelle WITH CHECK OPTION dans la définition de la vue.

*Exemple*

```
mysql> CREATE OR REPLACE VIEW rubrique_db AS
    -> SELECT * FROM rubrique WHERE id_parent = 1
    -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM rubrique_db;
+----+--------+-----------+
| id | titre  | id_parent |
+----+--------+-----------+
|  5 | MySQL  |         1 |
|  6 | Oracle |         1 |
+----+--------+-----------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM rubrique_db WHERE id_parent = 2;
Empty set (0.01 sec)

mysql> INSERT INTO rubrique_db(titre,id_parent) VALUES('SQLite',1);
Query OK, 1 row affected (0.01 sec)

mysql> INSERT INTO rubrique_db(titre,id_parent) VALUES('Langages',4);
ERROR 1369 (HY000): CHECK OPTION failed 'eni.rubrique_db'
```

Pour modifier la définition d'une vue, il est possible d'utiliser l'ordre SQL CREATE OR REPLACE VIEW ou l'ordre ALTER VIEW.

*Syntaxe simplifiée*

```
ALTER VIEW nom_vue [(nom_colonne[,...])]
AS requête_SELECT
[WITH CHECK OPTION]
```

L'ordre SQL `DROP VIEW` permet de supprimer une vue.

*Syntaxe*

```
DROP VIEW [IF EXISTS] nom_vue[,...]
```

---

L'ordre SQL `SHOW CREATE VIEW` permet d'afficher la définition d'une vue sous la forme d'un ordre `CREATE VIEW` (cf. Obtenir des informations sur les bases de données dans ce chapitre). L'ordre SQL `SHOW TABLES` affiche la liste des tables et des vues d'une base de données. L'ordre `DESC` peut être utilisé pour afficher la structure d'une vue.

---

# Obtenir des informations sur les bases de données

## 1. La commande SHOW

La commande SHOW propose de nombreuses variantes qui permettent d'afficher des informations sur les bases de données, les tables, les vues, etc.

*Variante 1 : afficher une liste d'objets*

```
SHOW DATABASES [condition]SHOW TABLES [FROM nom_base] [condition]SHOW TRIGGERS [FROM nom_base] [condition]
```

*Variante 2 : afficher la liste des colonnes ou des index d'une table*

```
SHOW COLUMNS FROM nom_table [FROM nom_base] [condition]SHOW INDEX FROM nom_table [FROM nom_base]
```

*Variante 3 : afficher l'ordre de création d'un objet*

```
SHOW CREATE {DATABASE | SCHEMA} nom_base
SHOW CREATE FUNCTION nom_fonction
SHOW CREATE PROCEDURE nom_procédure
SHOW CREATE TABLE nom_table
SHOW CREATE TRIGGER nom_trigger
SHOW CREATE VIEW nom_vue
```

*Variante 4 : afficher une liste d'objets avec quelques caractéristiques*

```
SHOW FUNCTION STATUS [condition]SHOW PROCEDURE STATUS [condition]SHOW TABLE STATUS [FROM nom_base] [condition]
```

*Avec*

```
condition = LIKE 'modèle' | WHERE expression
```

Le résultat de la commande SHOW est affiché sous la forme de lignes et de colonnes, comme le résultat d'une requête SELECT. La plupart des langages qui accèdent à MySQL permettent de traiter le résultat de la commande SHOW de la même manière que le résultat d'une requête SELECT.

La clause optionnelle condition permet de filtrer le résultat de la commande SHOW, soit à l'aide d'une expression LIKE qui s'applique implicitement au nom de l'objet, soit à l'aide d'une clause WHERE qui peut s'appliquer à n'importe quelle colonne du résultat.

*Exemple*

```
mysql> SHOW COLUMNS FROM collection;
+----------+--------------+------+-----+---------+----------------+
| Field    | Type         | Null | Key | Default | Extra          |
+----------+--------------+------+-----+---------+----------------+
| id       | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom      | varchar(25)  | NO   | UNI |         |                |
| prix_ht  | decimal(5,2) | YES  |     | 20.00   |                |
| frais_ht | decimal(5,2) | YES  |     | NULL    |                |
+----------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> SHOW CREATE TABLE collection;
+------------+---------------------------------------------------+
| Table      | Create Table                                      |
+------------+---------------------------------------------------+
| collection | CREATE TABLE `collection` (                       |
|            | `id` int(11) NOT NULL auto_increment,             |
|            | `nom` varchar(25) NOT NULL    ,                   |
|            | `prix_ht` decimal(5,2) default '20.00',           |
|            | `frais_ht` decimal(5,2) default NULL,             |
|            | PRIMARY KEY  (`id`),                              |
|            | UNIQUE KEY `nom` (`nom`)                          |
|            | ) ENGINE=MyISAM AUTO_INCREMENT=9                  |
|            |   DEFAULT CHARSET=latin1                          |
+------------+---------------------------------------------------+
1 row in set (0.01 sec)
```

## 2. La base de données INFORMATION_SCHEMA

Depuis la version 5, un serveur MySQL maintient une base de données nommée INFORMATION_SCHEMA qui contient un ensemble de tables qui permettent d'obtenir des informations sur les objets (tables, index, vues, etc.) contenus dans les différentes bases de données (ou "schémas" pour reprendre la nouvelle terminologie de la version 5). Cette base d'informations est parfois appelée "dictionnaire de données" ou "catalogue système". Il faut noter que la base de données INFORMATION_SCHEMA n'est pas une "vraie" base de données ; elle n'a ni

répertoire ni fichier sur le serveur.

Les tables de la base données INFORMATION_SCHEMA peuvent être interrogées comme n'importe quelle table pour obtenir des informations sur les objets stockés dans les différentes bases de données. Les tables suivantes permettent d'obtenir des informations sur les principaux objets d'une base de données :

COLUMNS

Informations sur les colonnes des tables.

COLUMN_PRIVILEGES

Informations sur les privilèges de niveau colonne des utilisateurs.

KEY_COLUMN_USAGE

Informations sur les colonnes des tables qui sont impliquées dans des contraintes.

ROUTINES

Informations sur les programmes stockés (cf. chapitre Techniques avancées avec MySQL - Développer des programmes stockés).

SCHEMATA

Informations sur les bases de données.

SCHEMA_PRIVILEGES

Informations sur les privilèges de niveau schéma (base de données) des utilisateurs.

TABLES

Informations sur les tables.

TABLE_CONSTRAINTS

Informations sur les contraintes des tables.

TABLE_PRIVILEGES

Informations sur les privilèges de niveau objet des utilisateurs.

TRIGGERS

Informations sur les triggers (cf. chapitre Techniques avancées avec MySQL - Développer des triggers).

USER_PRIVILEGES

Informations sur les privilèges globaux des utilisateurs.

VIEWS

Informations sur les vues.

---

Les différentes tables de la base de données INFORMATION_SCHEMA sont décrites en détail dans la documentation MySQL. Les ordres SHOW TABLES, SHOW COLUMNS, DESC peuvent être utilisés pour afficher des informations sur les tables de la base de données INFORMATION_SCHEMA.

---

*Exemple*

```
mysql> SELECT
    ->   column_name,
    ->   data_type,
    ->   column_default,
    ->   is_nullable,
    ->   column_key
    -> FROM
    ->   information_schema.columns
    -> WHERE
    ->   table_schema = 'eni'
    ->   AND table_name = 'collection'
    -> ORDER BY
    ->   ordinal_position;
+-------------+-----------+----------------+-------------+------------+
| column_name | data_type | column_default | is_nullable | column_key |
+-------------+-----------+----------------+-------------+------------+
| id          | int       | NULL           | NO          | PRI        |
| nom         | varchar   | NULL           | NO          | UNI        |
```

```
| prix_ht     | decimal   | 20.00          | YES         |            |
| frais_ht    | decimal   | NULL           | YES         |            |
+-------------+-----------+----------------+-------------+------------+
4 rows in set (0.01 sec)
```

## Exporter et importer une base de données

L'application cliente mysqldump permet d'exporter une base de données MySQL, sous la forme d'un fichier ("dump") contenant les ordres permettant de recréer la base de données. C'est un des moyens qui peut être utilisé pour sauvegarder une base de données.

*Syntaxe*

```
mysqldump [-h hôte] [-u utilisateur] [-p[mot_de_passe]] nom_base
```

*Avec*

`-h hôte`

Hôte auquel il faut se connecter (machine locale par défaut).

`-u utilisateur`

Nom d'utilisateur pour la connexion (nom de l'utilisateur courant du système d'exploitation par défaut).

`-p[mot_de_passe]`

Mot de passe pour la connexion (aucun mot de passe par défaut). S'il n'est pas donné sur la ligne de commande, il sera demandé de manière interactive, en saisie masquée. Si le mot de passe est spécifié dans la ligne de commande (ce qui n'est pas conseillé pour la sécurité), il ne doit pas y avoir d'espace après l'option -p.

*nom_base*

Nom de la base de données à exporter.

mysqldump affiche le résultat sur la sortie standard ; pour récupérer ce résultat dans un fichier, il faut utiliser une commande de redirection (> fichier).

*Exemple*

```
[root@xampp ~]# mysqldump -u eniadm -psecret eni > dump-base-eni.sql
```

*Résultat (extraits du fichier dump-base-eni.sql)*

```
-- MySQL dump 10.11
--
-- Host: localhost    Database: eni
-- ------------------------------------------------------
-- Server version       5.0.45

...

--
-- Table structure for table `rubrique`
--

DROP TABLE IF EXISTS `rubrique`;
CREATE TABLE `rubrique` (
  `id` int(11) NOT NULL auto_increment,
  `titre` varchar(20) NOT NULL,
  `id_parent` int(11) default NULL,
  PRIMARY KEY  (`id`),
  KEY `id_parent` (`id_parent`)
) ENGINE=MyISAM AUTO_INCREMENT=18 DEFAULT CHARSET=latin1;

--
-- Dumping data for table `rubrique`
--

LOCK TABLES `rubrique` WRITE;
/*!40000 ALTER TABLE `rubrique` DISABLE KEYS */;
INSERT INTO `rubrique` VALUES (1,'Base de données',NULL),(2,'Développement',NULL),(3,'Internet',NULL),(4,'Open Source',NULL),(5,'MySQL',1),(6,'Oracle',1),(7,'Langages',2),(8,'Méthode',2),(9,'HTML - XML',3),(10,'Conception Web',3),(11,'Sécurité',3),(12,'Système',4),(13,'Langages',4),(14,'Base de données',4),(16,'Certification',NULL),(17,'SQLite',1);
/*!40000 ALTER TABLE `rubrique` ENABLE KEYS */;
UNLOCK TABLES;

...

-- Dump completed on 2008-01-25 10:52:59
```

---

Attention s'il y a des mises à jour pendant la sauvegarde ! La sauvegarde risque de ne pas être cohérente du point de vue applicatif.

---

Un fichier d'export peut être importée dans une base de données qui existe déjà avec l'application cliente mysql soit en interactif (commande source - cf. chapitre Introduction à MySQL - Travailler avec MySQL) soit en batch, en utilisant une redirection de l'entrée standard : mysql ... < fichier.

*Exemple*

```
[root@xampp ~]# mysql -u root biblio < dump-base-eni.sql
[root@xampp ~]# mysql -u root biblio
Welcome to the MySQL monitor.  Commands end with ; or \g.
...

mysql> SELECT nom,prix_ht FROM collection;
+--------------------------+---------+
| nom                      | prix_ht |
+--------------------------+---------+
| Ressources Informatiques |   24.44 |
| TechNote                 |   10.48 |
| Les TP Informatiques     |   25.59 |
| Coffret Technique        |   46.45 |
| Solutions Informatiques  |   36.97 |
| ExpertIT                 |   20.00 |
+--------------------------+---------+
6 rows in set (0.01 sec)
```

# Grouper les données

Parfois, vous pouvez avoir besoin de calculer une valeur pour un niveau de regroupement :

- Chiffre d'affaires cumulé *par* région
- Salaire moyen *par* département

Pour ce genre d'interrogation, vous allez pouvoir utiliser des fonctions d'agrégat (`SUM`, `AVG`, etc.) et regrouper les données grâce à la clause `GROUP BY` ; en complément, le résultat final, après regroupement, peut être restreint avec la clause `HAVING`.

*Syntaxe :*

```
SELECT expression[,...] | *
FROM nom_table
[WHERE conditions]GROUP BY expression [ASC | DESC] [,...][HAVING conditions]
[ORDER BY expression [ASC | DESC][,...]]
[LIMIT [offset,] nombre_lignes]
```

La clause `GROUP BY` s'intercale entre les clauses `WHERE` et `ORDER BY` (si elles sont présentes). Elle spécifie les expressions utilisées pour effectuer le regroupement.

`expression` peut être :

- Une colonne
- Une expression basée sur des colonnes
- Un alias de colonne
- Un numéro correspondant à la position d'une expression de la clause `SELECT` (syntaxe déconseillée et obsolète)

Par défaut, le résultat de la requête est trié en ordre croissant sur les différentes expressions de la clause `GROUP BY` ; l'ordre du tri de chaque niveau de regroupement peut être défini explicitement avec les options `ASC` et `DESC`. Un tri complètement différent peut être spécifié grâce à la clause `ORDER BY`.

La plupart du temps, vous mettrez dans la clause `GROUP BY` toutes les expressions de la clause `SELECT` qui ne comportent pas de fonction d'agrégat. C'est le fonctionnement habituel pour respecter le SQL standard.

*Exemple*

```
mysql> -- Nombre de livres par collection.
mysql> SELECT id_collection,COUNT(*)
    -> FROM livre
    -> GROUP BY id_collection;
+---------------+----------+
| id_collection | COUNT(*) |
+---------------+----------+
|             1 |        4 |
|             2 |        3 |
|             3 |        1 |
|             4 |        1 |
+---------------+----------+
4 rows in set (0.00 sec)

mysql> -- Nombre de livres et nombre moyen de pages par collection
mysql> -- (avec le nom de la collection au lieu de l'identifiant).
mysql> SELECT
    ->   col.nom collection,
    ->   COUNT(liv.id) nb_livres,
    ->   ROUND(AVG(liv.nombre_pages)) nb_pages_moy
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
```

```
    -> GROUP BY
    ->   col.nom
    -> ORDER BY
    ->   nb_livres; -- tri sur le nombre de livres
+--------------------------+-----------+--------------+
| collection               | nb_livres | nb_pages_moy |
+--------------------------+-----------+--------------+
| Les TP Informatiques     |         1 |          272 |
| Coffret Technique        |         1 |          972 |
| TechNote                 |         3 |          182 |
| Ressources Informatiques |         4 |          486 |
+--------------------------+-----------+--------------+
4 rows in set (0.00 sec)

mysql> -- Nombre de livres par tranche d'année et collection.
mysql> SELECT
    ->   CASE
    ->     WHEN annee_parution IN (2004,2005)
    ->     THEN '2004-2005'
    ->     WHEN annee_parution IN (2006,2007)
    ->     THEN '2006-2007'
    ->   END tranche_annee,
    ->   col.nom collection,
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY -- utilisation des alias de colonne
    ->   tranche_annee,
    ->   collection;
+---------------+--------------------------+-----------+
| tranche_annee | collection               | nb_livres |
+---------------+--------------------------+-----------+
| 2004-2005     | Les TP Informatiques     |         1 |
| 2004-2005     | Ressources Informatiques |         2 |
| 2004-2005     | TechNote                 |         2 |
| 2006-2007     | Coffret Technique        |         1 |
| 2006-2007     | Ressources Informatiques |         2 |
| 2006-2007     | TechNote                 |         1 |
+---------------+--------------------------+-----------+
6 rows in set (0.01 sec)
```

Avec MySQL, vous pouvez avoir dans la clause SELECT des expressions qui n'utilisent pas de fonction d'agrégat et qui ne sont pas reprises dans la clause GROUP BY.

*Exemple :*

```
mysql> SELECT
    ->   col.nom collection, -- non présent dans le GROUP BY
    ->   col.prix_ht, -- non présent dans le GROUP BY
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.id;
+--------------------------+---------+-----------+
| collection               | prix_ht | nb_livres |
+--------------------------+---------+-----------+
| Ressources Informatiques |   24.44 |         4 |
| TechNote                 |   10.48 |         3 |
| Les TP Informatiques     |   25.59 |         1 |
| Coffret Technique        |   46.45 |         1 |
+--------------------------+---------+-----------+
4 rows in set (0.00 sec)
```

Cette possibilité de MySQL est intéressante et permet de simplifier l'écriture de la requête et d'améliorer les performances lorsque l'expression concernée est unique pour chaque groupe (c'est le cas sur l'exemple précédent où le nom et le prix de la collection sont uniques pour chaque identifiant de collection). Par contre, il est déconseillé d'utiliser

cette possibilité si l'expression concernée n'est pas unique dans le groupe ; MySQL va retourner n'importe quelle valeur pour le groupe et le résultat obtenu n'a en général pas vraiment de sens.

*Exemple*

```
mysql> SELECT
    ->   col.nom collection,
    ->   liv.annee_parution, -- pas unique dans le groupe !
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom;
+--------------------------+----------------+-----------+
| collection               | annee_parution | nb_livres |
+--------------------------+----------------+-----------+
| Coffret Technique        |           2006 |         1 |
| Les TP Informatiques     |           2004 |         1 |
| Ressources Informatiques |           2007 |         4 |
| TechNote                 |           2006 |         3 |
+--------------------------+----------------+-----------+
4 rows in set (0.00 sec)
```

Dans ce genre de situation, il peut être intéressant d'utiliser les fonctions MIN ou MAX pour obtenir une valeur spécifique "contrôlée" de l'expression ("le plus grand", "le plus petit", "le premier", "le dernier", etc.).

*Exemple*

```
mysql> SELECT
    ->   col.nom collection,
    ->   MIN(liv.annee_parution) premiere_parution,
    ->   COUNT(liv.id) nb_livres
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom;
+--------------------------+-------------------+-----------+
| collection               | premiere_parution | nb_livres |
+--------------------------+-------------------+-----------+
| Coffret Technique        |              2006 |         1 |
| Les TP Informatiques     |              2004 |         1 |
| Ressources Informatiques |              2004 |         4 |
| TechNote                 |              2005 |         3 |
+--------------------------+-------------------+-----------+
4 rows in set (0.00 sec)
```

Pour restreindre le résultat final, avec des conditions utilisant les fonctions d'agrégat, vous devez utiliser la clause HAVING. En effet, une clause WHERE ne peut pas comporter de condition qui utilise une fonction d'agrégat : c'est trop "tôt" dans le traitement de la requête ; les groupes n'ont pas encore été constitués.

Les conditions d'une clause HAVING sont semblables à celles d'une clause WHERE avec les particularités suivantes :

- elles peuvent utiliser des fonctions d'agrégat ;
- elles peuvent utiliser les alias définis dans la clause SELECT.

*Exemple :*

```
mysql> -- Afficher uniquement les collections qui ont
mysql> -- au moins deux livres.
mysql> SELECT
    ->   col.nom collection,
    ->   COUNT(liv.id) nb_livres,
    ->   ROUND(AVG(liv.nombre_pages)) nb_pages_moy
    -> FROM
    ->   livre liv JOIN collection col
```

```
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom
    -> HAVING
    ->   COUNT(liv.id) > 1;
+--------------------------+-----------+--------------+
| collection               | nb_livres | nb_pages_moy |
+--------------------------+-----------+--------------+
| Ressources Informatiques |         4 |          486 |
| TechNote                 |         3 |          182 |
+--------------------------+-----------+--------------+
2 rows in set (0.01 sec)

mysql> -- Afficher uniquement les collections qui ont
mysql> -- au moins deux livres et plus de 400 pages.
mysql> SELECT
    ->   col.nom collection,
    ->   COUNT(liv.id) nb_livres,
    ->   ROUND(AVG(liv.nombre_pages)) nb_pages_moy
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom
    -> HAVING -- utilisation des alias
    ->   nb_livres > 1
    ->   AND nb_pages_moy > 400;
+--------------------------+-----------+--------------+
| collection               | nb_livres | nb_pages_moy |
+--------------------------+-----------+--------------+
| Ressources Informatiques |         4 |          486 |
+--------------------------+-----------+--------------+
1 row in set (0.00 sec)
```

Pour des raisons de performance, la clause HAVING ne doit pas être utilisée pour filtrer des données qui pourraient l'être dans la clause WHERE.

# Utiliser des sous-requêtes

## 1. Introduction

Depuis la version 4.1, MySQL supporte l'utilisation de sous-requêtes.

Une sous-requête est une requête SELECT qui est utilisée à l'intérieur d'une autre requête.

Une sous-requête peut être utilisée :

- dans la clause WHERE d'une requête SELECT, UPDATE ou DELETE ;
- dans la clause SELECT d'une requête SELECT ;
- dans la clause FROM d'une requête SELECT ;
- comme valeur affectée à une colonne dans une requête INSERT ou UPDATE ;
- comme source de données d'une requête INSERT, à la place de la clause VALUES.

La sous-requête est toujours écrite entre parenthèses.

Sauf dans le cas de l'utilisation dans la clause FROM d'une requête SELECT, la sous-requête ne doit pas contenir de clause ORDER BY ; elle peut par contre contenir des jointures, ainsi que des clauses WHERE, GROUP BY, etc. Une sous-requête peut aussi elle-même contenir des sous-requêtes !

## 2. Sous-requête scalaire

Une sous-requête qui retourne une seule colonne et au plus une seule ligne est appelée sous-requête scalaire.

Une telle sous-requête peut être utilisée partout où une valeur (un opérande) est attendu :

- dans la clause WHERE d'une requête SELECT, INSERT, UPDATE ou DELETE ;
- dans la clause SELECT d'une requête SELECT ;
- comme valeur affectée à une colonne dans une requête INSERT ou UPDATE ;
- comme opérande d'une fonction ou d'une expression.

*Exemple*

```
mysql> -- Ecart entre le prix de la collection 1 et le prix moyen
mysql> -- des collections.
mysql> SELECT
    ->   ROUND(prix_ht - (SELECT AVG(prix_ht) FROM collection),2)
    -> FROM
    ->   collection
    -> WHERE
    ->   id = 1;
+----------------------------------------------------------+
| ROUND(prix_ht - (SELECT AVG(prix_ht) FROM collection),2) |
+----------------------------------------------------------+
|                                                    -2.88 |
+----------------------------------------------------------+
1 row in set (0.00 sec)

mysql> -- Prix TTC catalogue de l'article de code "RI410GORAA".
mysql> SELECT prix_ttc FROM catalogue WHERE code = 'RI410GORAA';
+----------+
```

```
| prix_ttc |
+----------+
|    35.00 |
+----------+
1 row in set (0.01 sec)

mysql> -- Affecter le prix TTC de la collection à l'article
mysql> -- du catalogue ayant le code "RI410GORAA".
mysql> UPDATE catalogue
    -> SET prix_ttc =
    ->   (SELECT prix_ht+ROUND(prix_ht*5.5/100)
    ->    FROM collection WHERE id = 1) -- sous-requête
    -> WHERE code = 'RI410GORAA';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> -- Vérifier.
mysql> SELECT prix_ttc FROM catalogue WHERE code = 'RI410GORAA';
+----------+
| prix_ttc |
+----------+
|    25.44 |
+----------+
1 row in set (0.00 sec)
```

Une sous-requête scalaire utilisée dans une clause WHERE est un cas particulier de l'utilisation plus générale d'une sous-requête dans une comparaison (cf. Comparaison avec une sous-requête).

## 3. Comparaison avec une sous-requête

Une sous-requête peut être utilisée comme expression dans la clause WHERE d'une autre requête :

```
{expression | sous-requête} opérateur {expression | sous-requête}
```

Les opérateurs habituels peuvent être utilisés : =, <, <=, >, >=, !=, IN, NOT IN, etc. Un opérateur spécial, EXISTS (et NOT EXISTS), peut aussi être utilisé.

Dans le cas des opérateurs scalaires (=, <, <=, >, >=, !=), la sous-requête doit retourner exactement une ligne ; le nombre d'expressions retournées doit de plus être égal au nombre d'expressions présentes dans l'autre membre de la comparaison.

*Exemple*

```
mysql> -- Collection la plus chère.
mysql> SELECT nom,prix_ht FROM collection
    -> WHERE prix_ht = (SELECT MAX(prix_ht) FROM collection);
+-------------------+---------+
| nom               | prix_ht |
+-------------------+---------+
| Coffret Technique |   46.45 |
+-------------------+---------+
1 row in set (0.01 sec)

mysql> -- Collections dont le prix est inférieur à la moyenne
mysql> -- du prix des collections.
mysql> SELECT nom,prix_ht FROM collection
    -> WHERE prix_ht <= (SELECT AVG(prix_ht) FROM collection);
+--------------------------+---------+
| nom                      | prix_ht |
+--------------------------+---------+
| Ressources Informatiques |   24.44 |
| TechNote                 |   10.48 |
| Les TP Informatiques     |   25.59 |
| ExpertIT                 |   20.00 |
+--------------------------+---------+
```

```
4 rows in set (0.00 sec)
```

Dans le cas des opérateurs IN, NOT IN, EXISTS et NOT EXISTS, la sous-requête peut retourner un nombre quelconque de lignes (avec la même règle sur le nombre d'expressions).

Le fonctionnement des opérateurs IN et NOT IN est le même qu'avec une liste explicite de valeurs.

*Exemple*

```
mysql> -- Titre des ouvrages des collections qui
mysql> -- n'ont qu'un ouvrage.
mysql> SELECT titre,id_collection
    -> FROM livre
    -> WHERE id_collection IN
    ->   -- La sous-requête donne la liste des identifiants
    ->   -- des collections qui n'ont qu'un ouvrage.
    ->   (SELECT id_collection FROM livre
    ->    GROUP BY id_collection HAVING COUNT(*) = 1);
+--------------------------------+---------------+
| titre                          | id_collection |
+--------------------------------+---------------+
| PHP et MySQL (versions 4 et 5) |             3 |
| MySQL 5 et PHP 5               |             4 |
+--------------------------------+---------------+
2 rows in set (0.01 sec)
```

L'opérateur EXISTS teste simplement l'existence d'un résultat dans la sous-requête ; la condition est évaluée à TRUE si la sous-requête retourne au moins une ligne, et à FALSE sinon. L'opérateur NOT EXISTS teste simplement la non existence d'un résultat dans la sous-requête ; la condition est évaluée à TRUE si la sous-requête ne retourne aucune ligne, et à FALSE sinon.

Les opérateurs EXISTS et NOT EXISTS sont généralement utilisés avec des sous-requêtes corrélées (cf. Sous-requête corrélée).

Il est possible d'utiliser les opérateurs scalaires (=, <, <=, >, >=, !=) avec une sous-requête qui retourne plusieurs lignes en les combinant avec les opérateurs ANY (ou son synonyme SOME) ou ALL. Ces opérateurs permettent de faire des interrogations du type "supérieur à une des valeurs de la sous-requête" ou "inférieur à toutes les valeurs de la sous-requête".

*Exemple*

```
mysql> -- Livres dont le nombre de pages est supérieur au nombre
mysql> -- de pages d'au moins un livre de la collection 1.
mysql> SELECT titre,nombre_pages,id_collection FROM livre
    -> WHERE nombre_pages > ANY
    ->   (SELECT nombre_pages FROM livre WHERE id_collection = 1);
+-------------------+--------------+---------------+
| titre             | nombre_pages | id_collection |
+-------------------+--------------+---------------+
| PHP 5.2           |          518 |             1 |
| Oracle 10g        |          489 |             1 |
| BusinessObjects 6 |          470 |             1 |
| MySQL 5 et PHP 5  |          972 |             4 |
+-------------------+--------------+---------------+
4 rows in set (0.00 sec)

mysql> -- Livres dont le nombre de pages est supérieur au nombre
mysql> -- de pages de tous les livres de la collection 1.
mysql> SELECT titre,nombre_pages,id_collection FROM livre
    -> WHERE nombre_pages > ALL
    ->   (SELECT nombre_pages FROM livre WHERE id_collection = 1);
+------------------+--------------+---------------+
| titre            | nombre_pages | id_collection |
+------------------+--------------+---------------+
| MySQL 5 et PHP 5 |          972 |             4 |
+------------------+--------------+---------------+
1 row in set (0.01 sec)
```

Pour en savoir plus sur cette syntaxe, consultez la documentation MySQL.

Attention à l'utilisation de certains opérateurs (NOT IN ou > ALL par exemple) lorsque la sous-requête retourne une valeur NULL ; le résultat est vide.

## 4. Sous-requête corrélée

Une sous-requête est dite corrélée si elle fait référence à une colonne d'une table de la requête parent. Dans ce cas, la sous-requête est exécutée et évaluée pour chaque ligne de la requête parent, car la colonne référencée dans la table de la requête parent est susceptible de changer à chaque ligne.

Dans le cas d'une sous-requête "simple" (non corrélée), la sous-requête est exécutée une seule fois et son résultat est comparé avec chaque ligne de la requête parent.

Les sous-requêtes corrélées peuvent être utilisées dans la clause WHERE d'une requête SELECT, UPDATE ou DELETE, ou dans la clause SET d'une requête UPDATE (dans ce cas, la sous-requête doit aussi être scalaire).

*Exemple*

```
mysql> -- Livres qui ont plus d'un auteur (requête  EXISTS).
mysql> SELECT liv.titre FROM livre liv
    ->  WHERE EXISTS
    ->   (SELECT 1 FROM auteur_livre aul WHERE aul.id_livre = liv.id
    ->    GROUP BY aul.id_livre HAVING COUNT(*) > 1);
+------------------+
| titre            |
+------------------+
| MySQL 5 et PHP 5 |
+------------------+
1 row in set (0.01 sec)

mysql> -- Livres qui ont plus d'un auteur (comparaison avec une
mysql> -- sous-requête scalaire).
mysql> SELECT liv.titre FROM livre liv
    -> WHERE
    ->   (SELECT COUNT(*) FROM auteur_livre aul
    ->    WHERE aul.id_livre = liv.id) > 1;
+------------------+
| titre            |
+------------------+
| MySQL 5 et PHP 5 |
+------------------+
1 row in set (0.00 sec)

mysql> -- Sous-rubriques de la rubrique 2 qui n'ont
mysql> -- pas de livre affecté.
mysql> SELECT rub.titre FROM rubrique rub
    -> WHERE
    ->   rub.id_parent = 2
    ->   AND NOT EXISTS
    ->     (SELECT 1 FROM rubrique_livre rul
    ->      WHERE rul.id_rubrique = rub.id);
+---------+
| titre   |
+---------+
| Méthode |
+---------+
1 row in set (0.00 sec)

mysql> -- Supprimer les sous-rubriques de la rubrique 2
mysql> -- qui n'ont pas de livre affecté.
mysql>  DELETE FROM rubrique
    -> WHERE
    ->   id_parent = 2
    ->   AND NOT EXISTS
    ->     (SELECT 1 FROM rubrique_livre rul
    ->      WHERE rul.id_rubrique = rubrique.id);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> -- Collections qui ont au moins un livre de plus
mysql> -- de 500 pages.
mysql> SELECT id,nom,frais_ht FROM collection col
    -> WHERE EXISTS
    ->   (SELECT 1 FROM livre liv
    ->    WHERE liv.id_collection = col.id
    ->          AND liv.nombre_pages > 500);
+----+-------------------------+----------+
| id | nom                     | frais_ht |
+----+-------------------------+----------+
|  1 | Ressources Informatiques |     1.50 |
|  4 | Coffret Technique       |     2.00 |
+----+-------------------------+----------+
2 rows in set (0.00 sec)

mysql> -- Augmenter de 25 centimes les frais pour les
mysql> -- collections qui ont au moins un livre de plus
mysql> -- de 500 pages.
mysql>  UPDATE collection col SET col.frais_ht = col.frais_ht + 0.25
    -> WHERE EXISTS
    ->   (SELECT 1 FROM livre liv
    ->    WHERE liv.id_collection = col.id
    ->          AND liv.nombre_pages > 500);
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> -- Vérifier.
mysql> SELECT id,nom,frais_ht FROM collection WHERE id IN (1,4);
+----+-------------------------+----------+
| id | nom                     | frais_ht |
+----+-------------------------+----------+
|  1 | Ressources Informatiques |     1.75 |
|  4 | Coffret Technique       |     2.25 |
+----+-------------------------+----------+
2 rows in set (0.00 sec)

mysql> -- Collections qui n'ont pas de frais.
mysql> SELECT * FROM collection WHERE frais_ht IS NULL;
+----+----------+---------+----------+
| id | nom      | prix_ht | frais_ht |
+----+----------+---------+----------+
|  2 | TechNote |   10.48 |     NULL |
|  6 | ExpertIT |   20.00 |     NULL |
+----+----------+---------+----------+
2 rows in set (0.00 sec)

mysql> -- Calculer les frais des collections qui n'en ont pas.
mysql> -- Formule = 0,4 centime x le nombre de pages moyen des
mysql> --                         livres de la collection
mysql> UPDATE collection col
    -> SET col.frais_ht =
    ->   (SELECT ROUND(AVG(nombre_pages)/250,2) FROM livre liv
    ->    WHERE liv.id_collection = col.id)
    -> WHERE col.frais_ht IS NULL;
Query OK, 1 row affected, 1 warning (0.00 sec)
Rows matched: 2  Changed: 1  Warnings: 0

mysql> -- Vérifier (la collection 6 n'a pas de livre, donc
mysql> -- les frais sont inchangés).
mysql> SELECT * FROM collection WHERE id IN (2,6);
+----+----------+---------+----------+
| id | nom      | prix_ht | frais_ht |
+----+----------+---------+----------+
|  2 | TechNote |   10.48 |     0.73 |
|  6 | ExpertIT |   20.00 |     NULL |
+----+----------+---------+----------+
2 rows in set (0.00 sec)
```

Dans une requête DELETE, le nom de table spécifié dans la clause FROM ne peut pas comporter d'alias. Pour référencer

une colonne de la table supprimée dans la sous-requête corrélée, il faut utiliser le nom de la table.

Une requête UPDATE ou DELETE avec sous-requête corrélée peut très souvent être remplacée par une requête UPDATE ou DELETE multi-tables.

## 5. Sous-requête dans la clause FROM

Une sous-requête placée dans la clause FROM d'une requête SELECT est utilisée comme source de données ; on parle parfois de "table dérivée" ou de "vue en ligne".

*Syntaxe*

```
(sous-requête) [AS] alias
```

Une sous-requête utilisée en clause FROM doit obligatoirement avoir un alias pour "nommer" la sous-requête. Par ailleurs, les expressions sélectionnées dans la sous-requête doivent avoir des noms différents. Enfin, une sous-requête utilisée en clause FROM peut avoir une clause ORDER BY.

*Exemple*

```
mysql> -- Afficher le nom de la collection et le premier livre
mysql> -- de la collection (ordre alphabétique du titre).
mysql> SELECT col.nom,liv.titre
    -> FROM
    ->   collection col
    ->     JOIN
    ->   (-- premier titre de la collection (ordre alphabétique)
    ->   SELECT id_collection,MIN(titre) titre
    ->   FROM livre GROUP BY id_collection
    ->   ) liv
    ->     ON (col.id = liv.id_collection);
+--------------------------+--------------------------------+
| nom                      | titre                          |
+--------------------------+--------------------------------+
| Ressources Informatiques | BusinessObjects 6              |
| TechNote                 | Oracle 10g                     |
| Les TP Informatiques     | PHP et MySQL (versions 4 et 5) |
| Coffret Technique        | MySQL 5 et PHP 5               |
+--------------------------+--------------------------------+
4 rows in set (0.00 sec)
```

## 6. Insérer des lignes à l'aide d'une sous-requête

Une sous-requête peut être utilisée comme source de données dans une requête d'insertion.

*Syntaxe*

```
INSERT [IGNORE]
[INTO] nom_table [(nom_colonne,...)]
sous-requête
[ ON DUPLICATE KEY UPDATE nom_colonne=expression, ... ]
```

La sous-requête remplace la clause VALUES ; le reste de la syntaxe est inchangé. La sous-requête doit retourner autant d'expressions qu'il y a de colonnes à insérer.

*Exemple*

*Création d'une nouvelle table destinée à stocker des informations statistiques sur les collections :*

```
mysql> CREATE TABLE statistique_collection
    ->   (
    ->   collection VARCHAR(30) PRIMARY KEY,
    ->   nombre_livres INT,
    ->   premiere_parution YEAR(4)
    ->   );
Query OK, 0 rows affected (0.05 sec)
```

*Alimentation de la table :*

```
mysql> INSERT INTO statistique_collection
    ->   (collection,nombre_livres,premiere_parution)
    -> SELECT
    ->   col.nom,COUNT(liv.id),MIN(liv.annee_parution)
    -> FROM
    ->   livre liv JOIN collection col
    ->   ON (liv.id_collection = col.id)
    -> GROUP BY
    ->   col.nom
    -> ON DUPLICATE KEY UPDATE
    ->   nombre_livres = VALUES(nombre_livres),
    ->   premiere_parution = VALUES(premiere_parution);
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM statistique_collection;
+--------------------------+---------------+-------------------+
| collection               | nombre_livres | premiere_parution |
+--------------------------+---------------+-------------------+
| Coffret Technique        |             1 |              2006 |
| Les TP Informatiques     |             1 |              2004 |
| Ressources Informatiques |             4 |              2004 |
| TechNote                 |             3 |              2005 |
+--------------------------+---------------+-------------------+
4 rows in set (0.01 sec)
```

# Réunir le résultat de plusieurs requêtes

Depuis la version 4, MySQL supporte l'utilisation de l'opérateur ensembliste UNION qui permet de réunir le résultat de plusieurs requêtes.

*Syntaxe*

```
   requête_SELECTUNION [ALL | DISTINCT]
   requête_SELECT
[UNION ...]
[ORDER BY tri]
```

Les ordres SELECT doivent avoir le même nombre d'expressions et les expressions correspondantes doivent normalement avoir le même type (selon la documentation). Dans la pratique, si les expressions correspondantes ne sont pas du même type, il semble qu'elles sont converties en chaîne de caractères pour faire l'union.

Le titre des colonnes du résultat final est défini par la première requête.

Par défaut, toutes les lignes retournées sont distinctes ; le même résultat est obtenu en utilisant l'option DISTINCT. L'utilisation de l'option ALL permet de conserver toutes les lignes, même celles dupliquées.

Les ordres SELECT ne doivent pas contenir de clause ORDER BY ; ils peuvent par contre contenir des jointures, des sous-requêtes, ainsi que des clauses WHERE, GROUP BY, etc. Pour renforcer la lisibilité de la requête, les ordres SELECT peuvent être mis entre parenthèse.

Le résultat final de l'union peut être trié par une clause ORDER BY, avec la même syntaxe que pour un ordre SELECT simple (cf. chapitre Introduction à MySQL - Exécuter des requêtes SQL simples).

*Exemples*

```
mysql> SELECT titre FROM catalogue
    ->   UNION
    -> SELECT IFNULL(CONCAT(titre,' - ',sous_titre),titre)
    -> FROM livre WHERE id_collection = 1;
+--------------------------------------------------------------------+
| titre                                                              |
+--------------------------------------------------------------------+
| Oracle 10g - Administration                                        |
| PHP 5 - L'accès aux données                                        |
| PHP 5.2 - Développer un site Web dynamique et interactif           |
| BusinessObjects 6                                                  |
| MySQL 5 - Installation, mise en œuvre, administration et programmation |
+--------------------------------------------------------------------+
5 rows in set (0.00 sec)

mysql> SELECT titre FROM catalogue
    ->   UNION ALL
    -> SELECT IFNULL(CONCAT(titre,' - ',sous_titre),titre)
    -> FROM livre WHERE id_collection = 1;
+--------------------------------------------------------------------+
| titre                                                              |
+--------------------------------------------------------------------+
| Oracle 10g - Administration                                        |
| PHP 5 - L'accès aux données                                        |
| PHP 5.2 - Développer un site Web dynamique et interactif           |
| Oracle 10g - Administration                                        |
| BusinessObjects 6                                                  |
| MySQL 5 - Installation, mise en œuvre, administration et programmation |
+--------------------------------------------------------------------+
6 rows in set (0.00 sec)

mysql> SELECT titre FROM catalogue
    ->   UNION
    -> SELECT IFNULL(CONCAT(titre,' - ',sous_titre),titre)
    -> FROM livre WHERE id_collection = 1
    ->   ORDER BY titre;
+--------------------------------------------------------------------+
| titre                                                              |
+--------------------------------------------------------------------+
```

```
| BusinessObjects 6                                                      |
| MySQL 5 - Installation, mise en œuvre, administration et programmation |
| Oracle 10g - Administration                                            |
| PHP 5 - L'accès aux données                                            |
| PHP 5.2 - Développer un site Web dynamique et interactif               |
+------------------------------------------------------------------------+
5 rows in set (0.00 sec)
```

# Gérer les transactions et les accès concurrents

## 1. Définition

Dans la terminologie des bases de données relationnelles, une transaction est un ensemble d'ordres de mise à jour qui forme un tout indissociable du point de vue de la logique applicative. Les ordres de mise à jour d'une transaction ne peuvent être définitivement enregistrés dans la base de données que s'ils se sont tous exécutés sans erreur ; si un des ordres de mise à jour échoue, toutes les modifications déjà effectuées dans la transaction doivent être annulées. À la fin d'une transaction, la base de données est toujours dans un état cohérent du point de vue de la logique applicative.

À titre d'exemple, considérons une transaction de virement bancaire qui serait constituée de trois ordres de mise à jour :

- un premier UPDATE pour débiter le premier compte ;
- un deuxième UPDATE pour créditer le deuxième compte ;
- un INSERT pour enregistrer l'opération dans un journal.

Si le deuxième UPDATE échoue pour une raison ou pour une autre, il faut annuler le premier UPDATE et ne pas effectuer l'ordre INSERT.

## 2. Gérer les transactions

Par défaut, MySQL fonctionne dans un mode validation automatique (option AUTOCOMMIT égale à 1) : chaque modification effectuée est immédiatement et définitivement enregistrée dans la base de données, ce qui ne permet pas de gérer correctement les transactions.

Par ailleurs, la gestion des transactions est supportée uniquement pour les types de table InnoDB et BDB.

À partir du moment où vous utilisez une table qui supporte les transactions, vous pouvez gérer les transactions à l'aide des instructions suivantes :

```
START TRANSACTION | BEGIN [WORK]

SET AUTOCOMMIT = { 0 | 1 }

COMMIT [WORK]

ROLLBACK [WORK]
```

Les instructions START TRANSACTION ou BEGIN (WORK est optionnel) permettent de démarrer explicitement une nouvelle transaction. Il est conseillé d'utiliser de préférence l'instruction START TRANSACTION qui est conforme au standard SQL. Lorsqu'une transaction est démarrée explicitement de cette manière, la validation automatique est désactivée le temps de la transaction ; à la fin de la transaction, la validation automatique revient dans sont état précédent.

L'instruction SET AUTOCOMMIT permet d'activer (1) ou de désactiver (0) la validation automatique pour la connexion courante.

À partir du moment où la validation automatique est désactivée, il n'est pas nécessaire de démarrer explicitement les transactions ; une nouvelle transaction est implicitement démarrée à la fin de chaque transaction.

L'instruction COMMIT valide la transaction courante et enregistre définitivement les modifications dans la base de données. L'instruction ROLLBACK annule la transaction courante et toutes les modifications effectuées par la transaction. Dans les deux cas, le mot clé WORK est optionnel.

Pour illustrer le fonctionnement des transactions, nous allons travailler avec les tables commande et ligne_commande créées dans la chapitre Construire une base de données dans MySQL à l'aide des ordres SQL suivants :

```
CREATE TABLE commande
  (
  id INT PRIMARY KEY AUTO_INCREMENT,
  date_commande DATE
```

```
  )
  ENGINE innodb;
CREATE TABLE ligne_commande
  (
  id_commande INT,
  numero_ligne INT,
  article VARCHAR(50),
  quantite INT,
  prix_unitaire DECIMAL(8,2),
  PRIMARY KEY(id_commande,numero_ligne)
  )
  ENGINE innodb;
```

Cet exemple utilise des tables InnoDB. Pour permettre l'utilisation des tables InnoDB, n'oubliez pas de mettre en commentaire le paramètre `skip-innodb` dans le fichier de configuration de MySQL.

*Exemple de transaction annulée*

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO commande(date_commande) VALUES(NOW());
Query OK, 1 row affected (0.01 sec)

mysql> SET @id = LAST_INSERT_ID();
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,1,'PHP 5.2',1,25);
Query OK, 1 row affected (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM commande;
Empty set (0.01 sec)

mysql> SELECT * FROM ligne_commande;
Empty set (0.01 sec)
```

*Exemple de transaction validée*

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO commande(date_commande) VALUES(NOW());
Query OK, 1 row affected (0.01 sec)

mysql> SET @id = LAST_INSERT_ID();
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,1,'PHP 5.2',1,25);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM commande;
+----+---------------+
| id | date_commande |
+----+---------------+
|  3 | 2008-01-25    |
+----+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT * FROM ligne_commande;
+-------------+-------------+---------+----------+--------------+
| id_commande | numero_ligne | article | quantite | prix_unitaire |
+-------------+-------------+---------+----------+--------------+
|           3 |            1 | PHP 5.2 |        1 |        25.00 |
+-------------+-------------+---------+----------+--------------+
1 row in set (0.00 sec)
```

Les ordres du langage de définition de données (LDD - Data Definition Language en anglais), comme CREATE, ALTER, DROP, ne peuvent pas être annulés. Par ailleurs, ces ordres valident implicitement la transaction en cours. De même, activer la validation automatique (SET AUTOCOMMIT = 1) ou démarrer une transaction (START TRANSACTION) valide implicitement la transaction en cours.

## 3. Annuler une partie d'une transaction

Dans certaines transactions comportant plusieurs ordres de mise à jour, il peut être nécessaire d'annuler uniquement les dernières modifications.

Vous pouvez annuler les dernières modifications d'une transaction à l'aide des instructions suivantes :

```
SAVEPOINT nom
```

```
ROLLBACK [WORK] TO [SAVEPOINT] nom
```

```
RELEASE SAVEPOINT nom
```

L'instruction SAVEPOINT permet de définir un point de retour nommé dans la séquence des ordres de mise à jour de la transaction. Si un point de retour portant le même nom était déjà défini dans la transaction, il est remplacé par le nouveau.

L'instruction ROLLBACK TO (les mots clés WORK et SAVEPOINT sont optionnels) permet d'annuler toutes les modifications réalisées depuis le point de retour indiqué. Les points de retour définis après le point de retour spécifié dans le ROLLBACK TO sont supprimés ; le point de retour spécifié dans le ROLLBACK TO est par contre conservé. La transaction en elle-même est toujours ouverte et toutes les modifications réalisées avant le point de retour indiqué sont en attente.

L'instruction RELEASE SAVEPOINT permet de supprimer un point de retour.

Tous les points de retour d'une transaction sont supprimés lorsque la transaction se termine (COMMIT ou ROLLBACK).

*Exemple*

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO commande(date_commande) VALUES(NOW());
Query OK, 1 row affected (0.01 sec)

mysql> SET @id = LAST_INSERT_ID();
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,1,'MySQL 5',1,25);
Query OK, 1 row affected (0.00 sec)

mysql> SAVEPOINT p1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,2,'PHP 7',1,25);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> ROLLBACK TO SAVEPOINT p1;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO ligne_commande
    ->   (id_commande,numero_ligne,article,quantite,prix_unitaire)
    -> VALUES
    ->   (@id,2,'PHP 5.2',1,25);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT * FROM commande WHERE id = @id;
+----+---------------+
| id | date_commande |
+----+---------------+
|  4 | 2008-01-25    |
+----+---------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM ligne_commande WHERE id_commande = @id;
+-------------+-------------+---------+----------+---------------+
| id_commande | numero_ligne | article | quantite | prix_unitaire |
+-------------+-------------+---------+----------+---------------+
|           4 |            1 | MySQL 5 |        1 |         25.00 |
|           4 |            2 | PHP 5.2 |        1 |         25.00 |
+-------------+-------------+---------+----------+---------------+
2 rows in set (0.00 sec)
```

## 4. Concurrence d'accès et verrouillage

### a. Concurrence d'accès

Au cours d'une transaction mettant à jour des tables InnoDB, MySQL pose des verrous pour éviter des mises à jour concurrentes qui seraient incorrectes.

Lorsqu'une session effectue un UPDATE ou un DELETE sur une table InnoDB au cours d'une transaction, deux cas peuvent se produire :

- L'ordre utilise un index pour trouver les lignes à modifier ou supprimer ; dans ce cas, MySQL va poser un verrou exclusif de niveau ligne sur les lignes concernées. Dans ce cas, les autres sessions attendent lors d'une tentative de modification ou de suppression des lignes verrouillées ; par contre, elles peuvent modifier ou supprimer d'autres lignes non verrouillées, ou en insérer de nouvelles (sauf cas particulier).

- L'ordre n'utilise pas d'index pour trouver les lignes à modifier ou supprimer ; dans ce cas, MySQL va poser un verrou exclusif sur toutes les lignes de la table. Dans ce cas, les autres sessions attendent lors de toute tentative de mise à jour sur la table (INSERT, UPDATE ou DELETE).

Dans les deux cas, les autres sessions peuvent interroger la table avec un ordre SELECT, mais elles ne voient pas les modifications non validées des autres sessions (pour plus de détails voir ci-après les explications sur le niveau d'isolation).

Il est possible de poser un verrou sur des lignes lors de la lecture en ajoutant la clause FOR UPDATE à la fin de l'ordre SELECT. Si une ligne sélectionnée est déjà verrouillée par une autre transaction, le SELECT FOR UPDATE est mis en attente ; il ne s'exécutera et verrouillera les lignes que lorsque la transaction se termine. Au passage, le SELECT FOR UPDATE permet de garantir que la ligne retournée est bien la dernière ligne mise à jour dans la base de données et qu'aucune mise à jour non validée n'est en cours sur cette ligne. Il faut noter que le SELECT FOR UPDATE verrouille toutes les lignes de la table s'il ne peut pas utiliser d'index pour sélectionner les lignes.

 Utiliser l'instruction SELECT FOR UPDATE est généralement conseillé pour gérer proprement les situations où deux utilisateurs sont susceptibles de modifier simultanément la même ligne (notion de verrouillage "pessimiste").

*Exemple 1*

**Session 1**

```
mysql> -- pas de COMMIT automatique
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE ligne_commande
    -> SET quantite = 2
    -> WHERE id_commande = 3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.02 sec)
```

**Session 2**

```
mysql> -- pas de COMMIT automatique
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT quantite
    -> FROM ligne_commande
    -> WHERE id_commande = 3;
+----------+
| quantite |
+----------+
|        1 |
+----------+
1 row in set (0.01 sec)

mysql> UPDATE ligne_commande
    -> SET quantite = 3
    -> WHERE id_commande = 3;
blocage ...
... au bout de quelques secondes
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction

mysql> UPDATE ligne_commande
    -> SET quantite = 3
    -> WHERE id_commande = 3;
blocage ...

Query OK, 1 row affected (3.42 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

Comme le montre l’exemple ci-dessus, il y a un délai d’attente maximum lorsqu’un ordre est bloqué par un verrou posé par une autre transaction ; lorsque le délai est écoulé et que l’ordre est toujours bloqué, MySQL annule l’ordre en attente et retourne un message d’erreur :

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

*Exemple 2*

**Session 1**

```
mysql> UPDATE ligne_commande
    -> SET quantite = 4
    -> WHERE id_commande = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT quantite
    -> FROM ligne_commande
    -> WHERE id_commande = 3
    -> FOR UPDATE;
blocage ...
... au bout de quelques secondes
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

**Session 2**

```
mysql> SELECT quantite
    -> FROM ligne_commande
    -> WHERE id_commande = 3
    -> FOR UPDATE;
blocage ...

+----------+
| quantite |
+----------+
|        2 |
+----------+
1 row in set (1.92 sec)
```

En terme de niveau d'isolation des transactions, les tables InnoDB fonctionnent dans le mode REPEATABLE READ. Ce mode garantit que les SELECT exécutés au cours de la transaction seront cohérents ; les modifications, même validées, des autres transactions effectuées après la première lecture se seront pas vues.

En cas de besoin, le niveau d'isolation d'une transaction peut être modifié à l'aide de l'instruction SET TRANSACTION ISOLATION LEVEL (voir la documentation MySQL sur les tables InnoDB pour plus d'informations).

### b. Verrouiller des tables

Dans certaines situations, il peut être intéressant de verrouiller des tables pour limiter l'accès d'autres utilisateurs à ces tables.

Un verrou peut être obtenu en *lecture* ou en *écriture*.

Un verrou en *lecture* permet à la session qui a posé le verrou et à toute autre session de lire la table ; par contre, aucune session (y compris celle qui a posé le verrou) ne peut écrire dans la table.

Un verrou en *écriture* permet à la session qui a posé le verrou de lire et d'écrire dans la table ; les autres sessions ne peuvent ni lire, ni écrire.

Verrouiller et déverrouiller des tables s'effectue à l'aide des instructions LOCK TABLES et UNLOCK TABLES.

*Syntaxe*

```
LOCK TABLES nom_table [AS alias] {READ|WRITE}[,...]

UNLOCK TABLES
```

Quand vous utilisez LOCK TABLES, vous devez verrouiller toutes les tables que vous allez utiliser. Si vous utilisez des alias dans l'instruction LOCK TABLES, vous devez utiliser ces alias dans vos requêtes. Si vous utilisez une table à plusieurs reprises avec des alias différents, vous devez verrouiller plusieurs fois la table en mentionnant chaque alias. Si des verrous sont déjà posés par d'autres sessions, l'instruction LOCK TABLES attendra que les verrous en question soient libérés.

Les verrous posés par LOCK TABLES sont libérés dans les cas suivants :

- l'instruction UNLOCK TABLES est exécutée ;
- une nouvelle instruction LOCK TABLE est exécutée ;
- une transaction est démarrée ;
- la session se termine.

Normalement, vous n'avez pas besoin de verrouiller les tables. Les deux cas principaux d'utilisation du verrouillage de

tables sont les suivants :

- pour améliorer les performances lorsque beaucoup de mises à jour sont effectuées sur plusieurs tables MyISAM (la raison en est que la mise à jour des index sur disque est retardée jusqu'au déverrouillage des tables) ;
- pour simuler une transaction lors de la mise à jour de tables qui ne supportent pas les transactions.

 Le verrouillage de table doit être utilisé avec parcimonie car il limite fortement la concurrence d'accès.

### c. Verrou mortel

MySQL sait détecter les situations de verrou mortel où deux sessions se bloquent mutuellement ; dans ce cas, MySQL annule l'ordre à l'origine de l'apparition du verrou mortel et retourne une erreur :

```
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting
transaction
```

# Effectuer des recherches à l'aide des expressions régulières

MySQL permet de faire des recherches à l'aide d'expressions régulières. Les expressions régulières permettent de spécifier des modèles complexes pour la recherche sur les chaînes.

MySQL propose deux opérateurs de comparaison pour effectuer des recherches à l'aide d'expressions régulières : REGEXP et RLIKE (synonyme de REGEXP).

*Syntaxe*

```
expression [NOT] REGEXP modèle

expression [NOT] RLIKE modèle
```

modèle est une expression régulière qui décrit la structure de la chaîne recherchée.

Une expression régulière peut être spécifiée à l'aide des symboles suivants (conforme à la norme POSIX) :

| **Caractère spécial** | **Signification** |
|---|---|
| ^ | Si ^ est présent comme premier caractère du modèle, indique que la chaîne doit commencer par ce qui suit :^abc : doit commencer par abc. |
| $ | Si $ est présent comme dernier caractère du modèle, indique que la chaîne doit terminer par ce qui précède : xyz$ : doit terminer par xyz.<br>^abcxyz$ : doit commencer par abcxyz et se terminer par abcxyz (bref être égal à abcxyz !).<br>Un modèle ne comportant ni ^ ni $ indique que le modèle est recherché n'importe où à l'intérieur de la chaîne :abc : contient abc. |
| * | Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), peut être présente zéro, une ou plusieurs fois : ab*c accepte ac, abc, abbc ... |
| + | Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), doit être présente une ou plusieurs fois : ab+c accepte abc, abbc ..., mais refuse ac. |
| ? | Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), peut être présente zéro ou une fois : ab?c accepte ac et abc mais refuse abbc, abbbc ... |
| {x} {x,}{x,y} | Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), doit être présente exactement x fois ({x}) ou au minimum x fois ({x,}) ou entre x et y fois ({x,y}) : ab{2}c n'accepte que abbc.<br>ab{2,4}c accepte abbc, abbbc, et abbbbc mais refuse abc (manque un b) ou abbbbbc (un b de trop).<br>ab{2,}c accepte abbc, abbbc, abbbbc, ... mais refuse abc (manque un b). |
| (...) | Permet de marquer une séquence recherchée, typiquement avec les symboles relatifs au nombre d'occurrences : a(bc)*d accepte ad, abcd, abcbcd ... mais refuse abd ou acd (la séquence bc n'est pas trouvée). |
| | Recherche de x ou de y (généralement utilisé avec les |

<table>
<tr><td>x|y</td><td>parenthèses pour éviter toute confusion) : a(b|c)*d accepte ad, abd, acd, abcd, abbcd, accbd ... (en clair un nombre quelconque de b ou de c situés dans n'importe quel ordre entre un a et un d).</td></tr>
<tr><td>[...]</td><td>Permet de spécifier une série de caractères acceptés soit sous la forme c1c2...cn pour une liste exhaustive précise, soit sous la forme c1-c2 pour une plage de caractères, soit en mélangeant les deux :[abcd] accepte un caractère parmi abcd (équivalent à (a|b|c|d)).<br><br>[a-z] accepte un caractère compris entre a et z.<br><br>[123a-zA-z] accepte un caractère compris entre a et z ou compris en A et Z ou égal à 1 ou 2 ou 3.<br><br>[a-zA-z0-9] accepte un caractère compris entre a et z ou entre A et Z ou entre 0 et 9.<br><br>Une exclusion peut être spécifiée en mettant un ^ en premier caractère à l'intérieur des crochets :<br><br>[^0-9] refuse tout caractère compris entre 0-9.<br><br>[^abc] refuse les caractères a, b et c.<br><br>Compte tenu de sa signification particulière, le signe -, s'il est recherché en tant que tel, doit figurer en premier ou en dernier entre les crochets.</td></tr>
<tr><td>.</td><td>Indique <u>un</u> caractère quelconque :<br><br>a.b accepte toute séquence comportant un a suivi d'exactement un caractère quelconque suivi d'un b : axb, ayb, mais pas ab ni axyb.<br><br>a.{0,2}b accepte toute séquence comportant un a suivi de zéro à deux caractères quelconques suivi d'un b : ab, axb, , axyb, mais pas axyzb.</td></tr>
<tr><td>\</td><td>Permet d'échapper les caractères spéciaux (^.[$()|*+?{\) lorsque ceux-ci sont recherchés en tant que tel, <u>sauf</u> lorsqu'ils sont mentionnés entre crochets :<br><br>a*{2,4}b permet de rechercher les séquences commençant par un a suivi de 2 à 4 étoiles suivies d'un b (notez le *).<br><br>a[$*+]{2,4}b permet de rechercher les séquences commençant par un a suivi de 2 à 4 caractères pris parmi $, * et + suivis d'un b (pas besoin de \)</td></tr>
<tr><td>[::]</td><td>Classe de caractères (voir ci-dessous).</td></tr>
</table>

Les classes de caractères sont les suivantes :

`[:alnum:]`

Caractères alphanumériques

`[:alpha:]`

Caractères alphabétiques

`[:blank:]`

Caractères blancs

`[:cntrl:]`

Caractères de contrôle

`[:digit:]`

Chiffres

`[:graph:]`

Caractères graphiques

`[:lower:]`

Caractères alphabétiques minuscules

`[:print:]`

Caractères graphiques ou blancs

`[:punct:]`

Caractères de ponctuation

`[:space:]`

Espace, tabulation, nouvelle ligne, retour chariot

`[:upper:]`

Caractères alphabétiques majuscules

`[:xdigit:]`

Caractères des chiffres hexadécimaux

*Exemples*

```
mysql> -- Titres qui contiennent "php".
mysql> SELECT titre FROM livre
    -> WHERE titre REGEXP 'php';
+---------------------------------+
| titre                           |
+---------------------------------+
| PHP 5.2                         |
| PHP 5                           |
| PHP et MySQL (versions 4 et 5) |
| MySQL 5 et PHP 5                |
+---------------------------------+
4 rows in set (0.01 sec)

mysql> -- Titres qui commencent par "php".
mysql> SELECT titre FROM livre
    -> WHERE titre REGEXP '^php';
+---------------------------------+
| titre                           |
+---------------------------------+
| PHP 5.2                         |
| PHP 5                           |
| PHP et MySQL (versions 4 et 5) |
+---------------------------------+
3 rows in set (0.00 sec)

mysql> -- Titres qui commencent par "php" suivi
mysql> -- d'un espace et d'un chiffre.
mysql> SELECT titre FROM livre
    -> WHERE titre REGEXP '^php [1-9]';
+---------+
| titre   |
+---------+
```

```
| PHP 5.2 |
| PHP 5   |
+---------+
2 rows in set (0.00 sec)

mysql> -- Titres qui commencent par "php" suivi
mysql> -- d'un espace et d'un numéro de version
mysql> -- sous la forme x.y.
mysql> SELECT titre FROM livre
    -> WHERE titre REGEXP '^php [1-9]\.[0-9]';
+---------+
| titre   |
+---------+
| PHP 5.2 |
+---------+
1 row in set (0.00 sec)

mysql> -- Titres qui contiennent "mysql" et "php"
mysql> -- dans n'importe quel ordre.
mysql> SELECT titre FROM livre
    -> WHERE titre REGEXP '((mysql).*(php))|((php).*(mysql))';
+--------------------------------+
| titre                          |
+--------------------------------+
| PHP et MySQL (versions 4 et 5) |
| MySQL 5 et PHP 5               |
+--------------------------------+
2 rows in set (0.00 sec)
```

Les recherches avec les expressions régulières sont sensibles à la casse uniquement avec les chaînes binaires.

# Effectuer des recherches en texte intégral

## 1. Principes

MySQL permet d'effectuer des recherches de mots sur l'ensemble d'un texte (ou de plusieurs textes).

Pour utiliser cette fonctionnalité, il faut :

- créer un index spécial, de type FULLTEXT ;
- utiliser la fonction MATCH AGAINST dans les recherches.

Les index FULLTEXT peuvent être créés sur des colonnes de type CHAR, VARCHAR ou TEXT, mais uniquement sur des tables MyISAM.

## 2. Création de l'index FULLTEXT

Les index FULLTEXT peuvent être créés lors de la création initiale de la table (dans l'ordre CREATE TABLE) ou ultérieurement (par un ordre ALTER TABLE ou un ordre CREATE FULLTEXT INDEX).

Pour créer un index FULLTEXT dans un ordre CREATE TABLE ou ALTER TABLE, il faut utiliser une clause FULLTEXT similaire à la clause INDEX présentée dans le chapitre Construire une base de données dans MySQL. L'ordre CREATE FULLTEXT INDEX est une variante de l'ordre CREATE INDEX présenté dans le chapitre Construire une base de données dans MySQL.

*Syntaxe*

```
CREATE TABLE nom_table
(
spécification_colonnes,
FULLTEXT(nom_colonne[,...])
)

ALTER TABLE nom_table ADD FULLTEXT(nom_colonne[,...])

CREATE FULLTEXT INDEX nom_index ON nom_table(nom_colonne[,...])
```

*Exemple*

```
mysql> CREATE FULLTEXT INDEX ix_texte
    -> ON livre(titre,sous_titre,description);
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

## 3. Effectuer une recherche en texte intégral

### a. Recherche classique

La fonction MATCH AGAINST permet d'effectuer des recherches en texte intégral.

*Syntaxe*

```
MATCH(nom_colonne[,...]) AGAINST (expression [IN BOOLEAN MODE | WITH QUERY
EXPANSION])
```

La clause MATCH liste les colonnes dans lesquelles effectuer la recherche ; les colonnes spécifiées doivent correspondre exactement à la liste des colonnes d'un index FULLTEXT.

La clause AGAINST permet de spécifier l'expression recherchée.

Les options IN BOOLEAN MODE et WITH QUERY EXPANSION sont présentées par la suite.

La recherche est insensible à la casse, même avec des chaînes binaires.

La fonction MATCH AGAINST retourne un nombre positif qui donne le niveau de pertinence de l'enregistrement vis-à-vis de l'expression recherchée ; la fonction retourne 0 s'il n'y a pas de similarité.

Lorsque la fonction MATCH AGAINST est utilisée comme condition dans une clause WHERE, seules les lignes ayant une pertinence différente de zéro sont retournées, et les lignes retournées sont triées par défaut par pertinence décroissante (lignes les plus pertinentes en premier).

La fonction MATCH AGAINST peut être présente dans la clause SELECT afin de voir le "score" de pertinence de chaque ligne.

L'expression recherchée peut être un simple mot, ou une liste de mots. Par défaut, les mots de 3 lettres ou moins sont ignorés. De même, il existe une liste prédéfinie de mots exclus car considérés comme trop communs.

La pertinence est basée sur :

- le nombre de mots dans la ligne,
- le nombre de mots distincts dans la ligne,
- le nombre total de mots dans la liste,
- le nombre de lignes qui contiennent un mot en particulier.

Un mot recherché présent dans de nombreuses lignes aura un poids faible ou nul ; à l'inverse, un mot recherché présent dans peu de lignes aura un poids fort. Si un mot recherché est présent dans plus de la moitié (50%) des enregistrements trouvés, il est exclu.

Dans la pratique, les résultats sont d'autant plus pertinents que la table contient un grand nombre de lignes. Si la table contient peu de lignes, la distribution statistique des mots n'est sans doute pas pertinente, et les recherches risquent de donner des résultats étranges.

*Exemple*

```
mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('mysql');
+--------------------------------------------------------------------+
| titre                                                              |
+--------------------------------------------------------------------+
| MySQL 5 et PHP 5
  Maîtrisez les sites web dynamiques                                 |
| MySQL 5
  Installation, mise en œuvre, administration et programmation       |
| PHP 5
  L'accès aux données (MySQL, Oracle, SQL Server, SQLite...)         |
| PHP et MySQL (versions 4 et 5)
  Entraînez-vous à créer des applications professionnelles           |
+--------------------------------------------------------------------+
4 rows in set (0.01 sec)

mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('mysql,oracle')
    ->       AND id_collection = 1;
+--------------------------------------------------------------------+
| titre                                                              |
+--------------------------------------------------------------------+
| Oracle 10g
  Administration                                                     |
| MySQL 5
  Installation, mise en œuvre, administration et programmation       |
+--------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
```

```
    ->       AGAINST('mysql,oracle,installation')
    ->       AND id_collection = 1;
+-------------------------------------------------------------------+
| titre                                                             |
+-------------------------------------------------------------------+
| MySQL 5
  Installation, mise en œuvre, administration et programmation      |
| Oracle 10g
  Administration                                                    |
+-------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

## b. Recherche en mode booléen

Dans la clause AGAINST, l'option IN BOOLEAN MODE permet d'effectuer une recherche en mode booléen.

Dans ce cas, l'expression recherchée peut contenir les opérateurs suivants :

+

Indique que le mot qui suit doit être présent dans la ligne retournée.

-

Indique que le mot qui suit ne doit pas être présent dans la ligne retournée.

<

Diminue la pertinence du mot qui suit.

>

Augmente la pertinence du mot qui suit.

~

Indique que le mot qui suit aura une contribution négative à la pertinence (diminue la pertinence de la ligne mais ne l'exclut pas comme avec l'opérateur -).

()

Regroupe une liste de mots.

*

Caractère joker (doit apparaître à la fin du mot).

""

Recherche la phrase entre guillemets telle quelle.

Si l'expression ne contient aucun opérateur, la ligne doit contenir au moins un des mots recherchés ; la pertinence de la ligne est d'autant plus élevée qu'elle contient plus de mots recherchés.

Les recherches effectuées dans le mode booléen n'utilisent pas le seuil de 50% et ne trient pas automatiquement le résultat par ordre décroissant de pertinence. De plus, ces recherches peuvent fonctionner sans index FULLTEXT, mais sont alors moins performantes. Les colonnes spécifiées dans la clause MATCH n'ont pas besoin de correspondre exactement à la liste de colonnes d'un index FULLTEXT.

*Exemple*

```
mysql> -- MySQL et Oracle.
mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('+mysql +oracle' IN BOOLEAN MODE);
+-------------------------------------------------------------------+
| titre                                                             |
+-------------------------------------------------------------------+
```

```
| PHP 5
  L'accès aux données (MySQL, Oracle, SQL Server, SQLite...)        |
+-------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> -- MySQL mais pas Oracle.
mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('+mysql -oracle' IN BOOLEAN MODE);
+-------------------------------------------------------------------+
| titre                                                             |
+-------------------------------------------------------------------+
| MySQL 5
  Installation, mise en œuvre, administration et programmation      |
| PHP et MySQL (versions 4 et 5)
  Entraînez-vous à créer des applications professionnelles          |
| MySQL 5 et PHP 5
  Maîtrisez les sites web dynamiques                                |
+-------------------------------------------------------------------+
3 rows in set (0.00 sec)

mysql> -- Installation de MySQL ou d'Oracle.
mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('+installation +(oracle,mysql)' IN BOOLEAN MODE);
+-------------------------------------------------------------------+
| titre                                                             |
+-------------------------------------------------------------------+
| Oracle 10g
  Installation du serveur sous Windows/Linux - Oracle Net           |
| MySQL 5
  Installation, mise en œuvre, administration et programmation      |
+-------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

### c. Recherche avec extension de requête

Dans la clause `AGAINST`, l'option `WITH QUERY EXPANSION` permet d'effectuer une recherche avec extension de requête.

Dans ce mode, la recherche s'effectue en 2 passes :

- La première passe effectue la recherche de l'expression et identifie les mots les plus fréquents du résultat.
- La deuxième passe recherche alors les mots les plus fréquents trouvés dans le résultat de la première passe.

Ce mode permet d'étendre la recherche en aveugle à des termes qui s'avèrent être très souvent associés à un mot recherché.

Ce mode de recherche doit plutôt être réservé à des recherches qui portent sur une expression courte.

*Exemple*

```
mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('sauvegarde');
+-------------------------------------------------------------------+
| titre                                                             |
+-------------------------------------------------------------------+
| Oracle 10g
  Sauvegarde et restauration de la base de données avec RMAN        |
+-------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONCAT(titre,'\n  ',sous_titre) titre FROM livre
    -> WHERE MATCH(titre,sous_titre,description)
    ->       AGAINST('sauvegarde' WITH QUERY EXPANSION);
+-------------------------------------------------------------------+
| titre                                                             |
```

```
+------------------------------------------------------------------+
| Oracle 10g
  Sauvegarde et restauration de la base de données avec RMAN       |
| PHP 5
  L'accès aux données (MySQL, Oracle, SQL Server, SQLite...)       |
| Oracle 10g
  Installation du serveur sous Windows/Linux - Oracle Net          |
| Oracle 10g
  Administration                                                   |
+------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

Sur cet exemple, MySQL a trouvé le titre qui contient le mot "sauvegarde" puis a étendu la recherche aux mots associés, notamment "oracle".

## 4. Réglage de la recherche en texte intégral

Plusieurs variables peuvent être définies dans le fichier de configuration de MySQL afin de régler le fonctionnement de la recherche en texte intégral.

La taille minimale et maximale des mots à indexer sont définies dans les variables systèmes `ft_min_word_len` (4 par défaut) et `ft_max_word_len` (84 par défaut).

La liste prédéfinie de mots exclus peut être gérée par l'intermédiaire de la variable `ft_stopword_file`. Dans cette variable, vous pouvez indiquer le chemin d'un fichier qui contient votre liste de mots interdits pour les recherches en texte intégral. Dans ce fichier, vous pouvez saisir votre liste de mots comme vous le souhaitez en utilisant un caractère non alphanumérique comme délimiteur (espace, virgule, retour à la ligne). Pour désactiver le filtre de mots exclus, vous pouvez affecter une chaîne vide à la variable `ft_stopword_file`.

Si vous le souhaitez, vous pouvez modifier une ou plusieurs des variables précédentes avant de créer vos index `FULLTEXT`. Si un index `FULLTEXT` a déjà été créé, il faut le reconstruire pour qu'il prenne en compte les nouvelles valeurs. Vous pouvez reconstruire les index d'une table avec l'ordre SQL suivant :

```
REPAIR TABLE nom_table QUICK
```

# Développer des programmes stockés

## 1. Introduction

Les procédures stockées et les fonctions sont une des principales nouveautés de la version 5 de MySQL.

Un programme stocké est un ensemble d'ordres SQL et d'instructions procédurales (structures de contrôle, déclaration de variables, etc.) qui réalise une tâche spécifique et qui est enregistré avec un nom dans la base de données. Le programme stocké peut ensuite être appelé à partir de différents environnements de développement pour exécuter la tâche en question.

Utiliser des programmes stockés offre plusieurs avantages :

- Améliorer les performances
  - Le code est stocké dans la base de données et il y a moins d'échange entre le client et le serveur.
- Réutiliser du code
  - Le code stocké peut être appelé par d'autres programmes qui n'ont pas besoin d'implémenter de nouveau la logique applicative.
- Renforcer l'intégrité des données
  - Les règles de gestion peuvent être codées à un seul endroit, dans les programmes stockés. Si les applications clientes n'ont pas le droit d'accéder directement aux tables mais doivent passer par les programmes stockés, l'intégrité des données est garantie.

Il existe deux types de programmes stockés :

- les procédures ;
- les fonctions.

---

Les programmes stockés requièrent la table `proc` dans la base `mysql`. En cas de migration à partir d'une version antérieure à la 5, pensez à mettre jour vos tables de droit (voir la documentation MySQL pour le mode opératoire).

---

## 2. Gestion des droits

Les privilèges suivants sont nécessaires pour gérer les programmes stockés :

- `CREATE ROUTINE` pour créer un programme stocké ;
- `ALTER ROUTINE` pour modifier ou supprimer un programme stocké (automatiquement attribué au créateur d'un programme stocké).

Par ailleurs, pour créer un programme stocké, il faut avoir les privilèges adaptés sur les objets (tables, vues, etc.) manipulés par le programme.

Pour exécuter un programme stocké, l'utilisateur doit disposer du privilège `EXECUTE` sur le programme en question (attribué automatiquement au créateur du programme).

Par défaut, un programme stocké s'exécute avec les droits du propriétaire du programme stocké. Cela signifie qu'un utilisateur qui a le droit d'exécuter un programme stocké n'a pas besoin d'avoir des droits sur les objets (tables, vues, etc.) manipulés par le programme. Ce fonctionnement est intéressant en termes de gestion de droit : pour accéder aux objets manipulés par le programme stocké, l'utilisateur est obligé de passer par l'exécution du programme stocké qui peut implémenter toutes les règles de gestion ou de sécurité adéquates.

## 3. Gestion des programmes stockés

Les ordres SQL `CREATE PROCEDURE` et `CREATE FUNCTION` permettent de créer une procédure stockée ou une fonction stockée.

*Syntaxe*

```
CREATE PROCEDURE [nom_base.]nom_programme ([spécification_paramètre[,...]])
BEGIN
instructions;
END;

CREATE FUNCTION [nom_base.]nom_programme ([spécification_paramètre[,...]])
RETURNS type
BEGIN
instructions;
END;

spécification_paramètre =
[ IN | OUT | INOUT ] nom_paramètre type
```

`nom_base` désigne la base de données dans laquelle le programme stocké doit être défini. Par défaut, le programme est stocké dans la base de données courante.

`nom_programme` spécifie le nom du programme stocké. Ce nom doit respecter les règles de nommage des objets MySQL.

`type` désigne tout type de données MySQL valide.

La définition d'une fonction comporte une clause `RETURNS` qui permet de spécifier le type de données retourné par la fonction. Le code de la fonction contient aussi obligatoirement une instruction `RETURN` qui permet de définir la valeur retournée par la fonction. La clause `RETURNS` et l'instruction `RETURN` ne sont pas autorisés pour une procédure stockée.

Derrière le nom du programme, il est possible de définir une liste de paramètres. Si le programme n'utilise pas de paramètres, une liste vide `()` doit être spécifiée.

Un paramètre est défini par un mode de passation optionnel, un nom, et un type de données.

Les modes de passation autorisés sont `IN` (valeur par défaut), `OUT` et `INOUT` :

`IN`

Paramètre en entrée. Un paramètre `IN` permet au programme appelant de passer une valeur au programme stocké. Pour donner une valeur au paramètre, le programme appelant peut utiliser n'importe quelle expression (variable, expression littérale, etc.). Si le programme stocké modifie la valeur du paramètre, la modification n'est pas visible dans le programme appelant.

`OUT`

Paramètre en sortie. Un paramètre `OUT` permet au programme stocké de retourner une valeur au programme appelant. Le programme appelant doit obligatoirement utiliser une variable lors de l'appel pour récupérer la valeur retournée. La valeur initiale d'un paramètre `OUT` à l'intérieur du programme stocké est toujours `NULL`.

`INOUT`

Paramètre en entrée/sortie. Un paramètre `INOUT` est initialisé par l'appelant avec une valeur visible par le programme stocké. Ce dernier peut modifier la valeur du paramètre et la valeur modifiée est retournée au programme appelant. Le programme appelant doit obligatoirement utiliser une variable lors de l'appel pour récupérer la valeur retournée.

---

Seuls les paramètres `IN` sont autorisés pour les fonctions stockées, et le mode de passation ne doit pas être indiqué (même mettre `IN` génère une erreur).

---

Les instructions qui composent le code du programme sont incluses entre les mots clés `BEGIN` et `END`. Si le code du programme ne comporte qu'une seule instruction (cas assez rare !), les mots clés `BEGIN` et `END` peuvent être omis. Chaque instruction doit se terminer par un point-virgule. Le code du programme peut contenir des déclarations (variable, gestionnaire d'erreur), des affectations de valeurs à des variables, des structures de contrôle (boucle, conditions) et des ordres SQL.

Lorsque le programme stocké est créé avec l'interpréteur ligne de commande mysql, il ne faut pas que le point-virgule utilisé pour marquer la fin des instructions du code du programme soit interprété par mysql. Avant de soumettre la définition de la procédure, il est nécessaire d'utiliser la commande delimiter afin de modifier le délimiteur de fin d'instruction utilisé par le client MySQL.

*Exemples simples*

```
mysql> delimiter //
mysql>
mysql> CREATE PROCEDURE ps_creer_rubrique
    ->   (
    ->   -- Titre de la nouvelle rubrique.
    ->   IN p_titre VARCHAR(20),
    ->   -- Identifiant de la rubrique parent.
    ->   IN p_id_parent INT,
    ->   -- Identifiant de la nouvelle rubrique.
    ->   OUT p_id INT
    ->   )
    -> BEGIN
    ->   /*
    ->   ** Insérer la nouvelle rubrique et
    ->   ** récupérer l'identifiant affecté.
    ->   */
    ->   INSERT INTO rubrique (titre,id_parent)
    ->   VALUES (p_titre,p_id_parent);
    ->   SET p_id = LAST_INSERT_ID();
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE FUNCTION fs_titre_long
    ->   (
    ->   p_titre VARCHAR(100),
    ->   p_sous_titre VARCHAR(100)
    ->   )
    ->   RETURNS VARCHAR(210)
    -> BEGIN
    ->   RETURN CONCAT(p_titre,' - ',p_sous_titre);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
mysql>
```

Les différentes variantes de l'ordre SQL SHOW (cf. chapitre Construire une base de données dans MySQL - Obtenir des informations sur les bases de données) permettent d'afficher des informations sur les programmes stockés.

*Syntaxe*

```
SHOW CREATE {PROCEDURE | FUNCTION} nom_programme
SHOW {PROCEDURE | FUNCTION} STATUS [condition]
```

Les programmes stockés peuvent être supprimés à l'aide d'un ordre SQL DROP.

*Syntaxe*

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] nom_programme
```

## 4. Exécuter un programme stocké

L'ordre SQL CALL permet d'exécuter une procédure stockée.

*Syntaxe*

```
CALL [nom_base.]nom_procédure[([expression,[...]])]
```

Si la procédure stockée ne comporte pas de paramètre, elle peut être appelée sans parenthèse.

Le programme appelant doit utiliser des variables pour le passage des paramètres OU ou INOUT.

L'ordre CALL est utilisable dans un programme stocké pour appeler une procédure stockée.

*Exemple*

```
mysql> CALL ps_creer_rubrique('Système et réseau',NULL,@id);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM rubrique WHERE id = @id;
+----+-------------------+-----------+
| id | titre             | id_parent |
+----+-------------------+-----------+
| 18 | Système et réseau |      NULL |
+----+-------------------+-----------+
1 row in set (0.00 sec)
```

Une fonction stockée peut être appelée dans une expression comme une fonction prédéfinie.

*Exemple*

```
mysql> SELECT fs_titre_long(titre,sous_titre) FROM livre LIMIT 1;
+-------------------------------------------------------+
| fs_titre_long(titre,sous_titre)                       |
+-------------------------------------------------------+
| PHP 5.2 - Développer un site Web dynamique et interactif |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

Un programme stocké appartient à une base de données. Pour être appelé à partir d'une autre base de données, le nom du programme doit être préfixé par le nom de la base de données dans laquelle il est stocké. Par ailleurs, lorsque le programme stocké est appelé, il effectue un changement de base de données implicite, pour se positionner dans la base de données dans laquelle il est défini ; la base de données d'origine est automatiquement restaurée à la fin de l'exécution.

## 5. Structure du langage

### a. Bloc BEGIN END

Les mots clés BEGIN et END permettent de regrouper des instructions.

*Syntaxe*

```
[label:] BEGIN
   instructions;
END [label]
```

À l'intérieur du bloc, chaque instruction doit se terminer par un point-virgule. Un bloc BEGIN END peut être vide et ne contenir aucune instruction !

Un bloc BEGIN END peut contenir d'autres blocs BEGIN END ; dans ce cas, le bloc imbriqué doit se terminer par un point-virgule comme s'il s'agissait d'une instruction.

*Exemple*

```
BEGIN
  INSERT INTO rubrique (titre,id_parent)
  VALUES (p_titre,p_id_parent);
  SET p_id = LAST_INSERT_ID();
END;
```

Un bloc BEGIN END peut être "nommé" à l'aide d'un label situé devant le mot clé BEGIN et suffixé par le caractère deux points ; dans ce cas, le label doit être repris (sans le caractère deux point) sur le END final du bloc.

L'instruction LEAVE peut être utilisée pour sortir prématurément d'un bloc.

*Syntaxe*

```
LEAVE label
```

Pour pouvoir utiliser l'instruction LEAVE à l'intérieur d'un bloc BEGIN END, ce dernier doit être nommé à l'aide d'un label.

### b. Les variables

À l'intérieur d'un bloc, les variables locales utilisées par le programme doivent être déclarées à l'aide de l'instruction DECLARE.

*Syntaxe*

```
DECLARE nom_variable[,...] type [DEFAULT expression]
```

nom_variable est le nom donné à la variable ; ce nom doit respecter les règles de nommage de MySQL.

type est le type de données de la variable ; tout type de données MySQL valide peut être utilisé.

La clause optionnelle DEFAULT permet de donner une valeur par défaut (initiale) à la variable ; toute expression valide peut être utilisée. Si la clause est absente, la variable est initialisée à NULL.

L'instruction DECLARE peut comporter plusieurs noms de variables pour déclarer en une seule instruction plusieurs variables qui auront le même type de données et seront initialisées avec la même valeur.

Un bloc peut contenir plusieurs instructions DECLARE.

Une variable déclarée dans un bloc peut être utilisé dans ce bloc, et dans les blocs imbriqués du bloc. L'inverse n'est pas vrai : une variable déclarée dans un bloc imbriqué ne peut pas être utilisée dans le bloc parent.

Un bloc parent et un bloc imbriqué peuvent déclarer une variable portant le même nom mais ce sont deux variables différentes. Dans le bloc imbriqué, la variable du bloc parent ne peut pas être utilisée (elle n'est pas "visible" car "masquée" par la variable de même nom définie dans le bloc imbriqué). Déclarer ainsi deux variables portant le même nom n'est de toute façon pas une bonne idée de programmation.

Pour affecter une valeur à une variable, l'instruction SET peut être utilisée.

*Syntaxe*

```
SET nom_variable = expression [, nom_variable = expression ] [, ...]
```

Il est possible d'affecter des valeurs à plusieurs variables (même de types différents) en une seule instruction SET.

*Exemple*

```
BEGIN
  DECLARE v_id INT DEFAULT 0;
  DECLARE v_titre VARCHAR(20);
  DECLARE v_date DATE;
  SET v_date = CURRENT_DATE();
END;
```

Une valeur peut aussi être affectée à une variable à l'aide de l'ordre SQL SELECT INTO (voir ci-après).

### c. Intégration d'ordres SQL

Pratiquement tous les ordres SQL du langage peuvent être intégrés dans le code d'un programme stocké.

Les ordres SQL suivants sont interdits dans les programmes stockés (procédures ou fonctions) :

- {LOCK | UNLOCK} TABLES
- LOAD DATA et LOAD TABLE
- USE

En complément, pour les fonctions, les ordres SQL suivants sont interdits :

- instructions qui font explicitement ou implicitement un COMMIT ou un ROLLBACK ;
- instructions qui retournent un résultat directement au client (SELECT sans INTO, SHOW, etc.) ;
- modifier la table utilisée par l'ordre qui a déclenché l'appel de la fonction ;
- appeler une procédure qui utilise un ordre interdit dans les fonctions.

Un programme stocké ne peut pas utiliser l'instruction USE pour changer de base de données. Par contre, il peut référencer les objets (tables, vues, programmes stockés, etc.) d'une autre base de données en préfixant le nom des objets par le nom de la base de données dans laquelle ils sont définis.

Les ordres SQL imbriqués dans le code d'un programme stocké respectent la syntaxe habituelle et peuvent utiliser les paramètres et variables locales du programme.

Lorsqu'un ordre SQL SELECT est utilisé tel quel à l'intérieur d'une procédure stockée, le résultat de la requête est envoyé directement au programme appelant. Si la procédure contient plusieurs ordres SELECT de ce type, le programme appelant doit supporter la récupération de plusieurs ensembles de résultats.

L'utilisation directe d'un ordre SQL SELECT (ou de tout ordre qui retourne un résultat comme SHOW par exemple) est interdit pour une fonction stockée.

À l'intérieur d'un programme stocké, l'ordre SQL SELECT INTO peut être utilisé pour récupérer le résultat de la requête dans des variables.

*Syntaxe*

```
SELECT expression[,...] INTO nom_variable[,...] FROM ...
```

La clause INTO doit contenir autant de variables qu'il y a d'expressions dans le SELECT.

Avec cette syntaxe, la requête SELECT doit retourner au plus une ligne. Toutes les clauses habituelles de la requête SELECT peuvent être utilisées.

*Exemple*

```
CREATE PROCEDURE ps_creer_rubrique
  (
  -- Titre de la nouvelle rubrique.
  IN p_titre VARCHAR(20),
  -- Titre de la rubrique parent.
  IN p_titre_parent VARCHAR(20),
  -- Identifiant de la nouvelle rubrique.
  OUT p_id INT
  )
BEGIN
  -- Identifiant de la rubrique parent.
  DECLARE v_id_parent INT;
  -- Lire l'identifiant de la rubrique parent.
  SELECT id INTO v_id_parent
  FROM rubrique WHERE titre = p_titre_parent;
  -- Insérer la nouvelle rubrique.
  INSERT INTO rubrique (titre,id_parent)
  VALUES (p_titre,v_id_parent);
  -- Lire l'identifiant de la nouvelle rubrique.
  SET p_id = LAST_INSERT_ID();
END;
```

### d. Les structures de contrôle

MySQL supporte plusieurs structures de contrôle :

- Contrôle conditionnel : IF et CASE
- Contrôle itératif : LOOP, WHILE et REPEAT

Dans les descriptions de syntaxe qui suivent, instructions désigne une ou plusieurs instructions, ou un bloc

d'instructions.

Les structures de contrôle peuvent être imbriquées.

## IF

*Syntaxe*

```
IF condition THEN instructions
[ELSEIF condition THEN instructions]
...
[ELSE instructions]
END IF
```

Les conditions des clauses IF et ELSEIF sont testées séquentiellement. Si la condition est vraie alors les instructions associées de la clause THEN sont exécutées ; si aucune des conditions n'est vraie, les instructions définies dans la clause ELSE sont exécutées.

*Exemple*

```
CREATE PROCEDURE ps_calculer_frais_collection(p_id INT)
BEGIN
  -- Prix de la collection.
  DECLARE v_prix_ht INT;
  -- Frais de la collection.
  DECLARE v_frais_ht INT;
  -- Lire le prix de la collection.
  SELECT prix_ht INTO v_prix_ht
  FROM collection WHERE id = p_id;
  -- Calculer les frais de la collection.
  IF v_prix_ht <= 15
  THEN
    SET v_frais_ht = 1;
  ELSEIF v_prix_ht <= 40
  THEN
    SET v_frais_ht = 1.5;
  ELSE
    SET v_frais_ht = 2;
  END IF;
  -- Mettre à jour les frais dans la table.
  UPDATE collection
  SET frais_ht = v_frais_ht
  WHERE id = p_id;
END;
```

## CASE

*Syntaxe 1*

```
CASE expression_test
   WHEN expression THEN instructions
   [WHEN expression THEN instructions]
   ...
   [ELSE instructions]
END CASE
```

*Syntaxe 2*

```
CASE
   WHEN condition THEN instructions
   [WHEN condition THEN instructions]
   ...
   [ELSE instructions]
END CASE
```

Dans la première syntaxe, expression_test est une expression qui est évaluée une fois et dont la valeur est comparée séquentiellement avec les expressions des clauses WHEN. Si expression_test est égale à l'expression de la clause WHEN, alors les instructions associées de la clause THEN sont exécutées ; si aucune égalité n'est trouvée, les

instructions définies dans la clause ELSE sont exécutées.

Dans la deuxième syntaxe, les conditions des clauses WHEN sont testées séquentiellement. Si la condition est vraie alors les instructions associées de la clause THEN sont exécutées ; si aucune des conditions n'est vraie, les instructions définies dans la clause ELSE sont exécutées.

Dans les deux cas, la clause ELSE est optionnelle mais si cette clause est omise et qu'aucune égalité ou condition n'est vraie, une erreur est levée :

```
ERROR 1339 (20000): Case not found for CASE statement
```

Pour éviter cette erreur, vous pouvez définir une clause ELSE qui ne fait rien grâce à un bloc BEGIN END vide.

*Exemple*

```
BEGIN
...
  -- Calculer les frais de la collection.
  CASE
    WHEN v_prix_ht <= 15
    THEN
      SET v_frais_ht = 1;
    WHEN v_prix_ht <= 40
    THEN
      SET v_frais_ht = 1.5;
    ELSE
      SET v_frais_ht = 2;
  END CASE;
...
END;
```

## LOOP

*Syntaxe*

```
[label:] LOOP
   instructions
END LOOP [label]
```

La structure LOOP implémente une simple boucle qui permet l'exécution répétée d'une ou de plusieurs instructions. Un label peut être utilisé pour nommer la boucle ; dans ce cas, le label doit être repris dans le END LOOP. Nommer la boucle avec un label est obligatoire pour pouvoir utiliser les instructions LEAVE et ITERATE (voir ci-dessous).

Pour sortir de la boucle, il faut utiliser l'instruction LEAVE présentée précédemment.

*Exemple*

```
BEGIN
  DECLARE v_indice INT DEFAULT 0;
  boucle: LOOP
    SET v_indice = v_indice + 1;
    IF v_indice = 10
    THEN
      LEAVE boucle;
    END IF;
  END LOOP boucle;
END;
```


Attention aux conditions NULL. Une condition NULL (résultant par exemple d'un test sur une variable non initialisée) n'est pas TRUE. Si la condition testée dans le IF ne devient jamais vraie, la boucle s'exécute sans fin.

L'instruction ITERATE peut être utilisée pour passer immédiatement à l'itération suivante.

*Syntaxe*

```
ITERATE label
```

#### REPEAT

*Syntaxe*

```
[label:] REPEAT
   instructions
UNTIL condition
END REPEAT [label]
```

Les instructions à l'intérieur de la boucle sont exécutées jusqu'à ce que la condition de la clause UNTIL soit vraie (TRUE). Un label peut être utilisé pour nommer la boucle ; dans ce cas, le label doit être repris dans le ENDREPEAT.

*Exemple*

```
BEGIN
  DECLARE v_indice INT DEFAULT 0;
  REPEAT
    SET v_indice = v_indice + 1;
  UNTIL v_indice = 10
  END REPEAT;
END;
```

Les instructions LEAVE et ITERATE peuvent être utilisées à l'intérieur de la boucle, qui doit dans ce cas être forcément nommée à l'aide d'un label.

 Si la condition de la clause UNTIL ne devient jamais vraie (reste FALSE ou NULL), la boucle s'exécute sans fin.

#### WHILE

*Syntaxe*

```
[label:] WHILE condition DO
   instructions
END WHILE [label]
```

Les instructions à l'intérieur de la boucle sont exécutées tant que la condition de la clause WHILE est vraie. Un label peut être utilisé pour nommer la boucle ; dans ce cas, le label doit être repris dans le END WHILE.

*Exemple*

```
BEGIN
  DECLARE v_indice INT DEFAULT 0;
  WHILE v_indice < 10 DO
    SET v_indice = v_indice + 1;
  END WHILE;
END;
```

Les instructions LEAVE et ITERATE peuvent être utilisées à l'intérieur de la boucle, qui doit dans ce cas être forcément nommée à l'aide d'un label.

 Si la condition de la clause WHILE est NULL, la boucle ne s'exécute pas.

### e. La gestion des erreurs

La gestion des erreurs d'exécution s'effectue à l'aide de conditions et de gestionnaires (*handler*).

Une condition est en fait un moyen de nommer des erreurs spécifiques. Une condition peut être déclarée à l'aide de l'instruction DECLARE.

*Syntaxe*

```
DECLARE nom_condition CONDITION FOR
    code_erreur_mysql
  | SQLSTATE [VALUE] 'code_erreur_sql'
```

nom_condition est le nom donné à la condition.

Une condition peut être associée à un code d'erreur MySQL ou à un code d'erreur SQL (clause SQLSTATE).

Les codes d'erreurs SQL et MySQL sont décrits dans la documentation MySQL. Dans l'interface ligne de commande mysql, les erreurs sont affichées de la manière suivante :

```
ERROR code_erreur_mysql (code_erreur_sql) : message
```

Le code d'erreur MySQL est un nombre ; le code d'erreur SQL est une chaîne de 5 caractères.

Un gestionnaire permet de définir les instructions à exécuter lorsqu'une erreur se produit. Un gestionnaire peut être déclaré à l'aide de l'instruction DECLARE.

*Syntaxe*

```
DECLARE {CONTINUE|EXIT} HANDLER FOR condition [,...] instructions

condition =
    SQLSTATE 'code_erreur_sql'
  | code_erreur_mysql
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | nom_condition
```

instructions spécifie les instructions à exécuter lorsque la condition d'erreur se produit. Il peut s'agir d'une instruction unique ou d'un bloc d'instructions (délimité par les mots clés BEGIN et END). Un gestionnaire peut être associé à plusieurs conditions d'erreur.

Les valeurs possibles pour la condition d'erreur sont les suivantes :

SQLSTATE 'code_erreur_sql'

Un code d'erreur SQL.

code_erreur_mysql

Un code d'erreur MySQL.

SQLWARNING

Un code d'erreur SQL qui commence par 01.

NOT FOUND

Un code d'erreur SQL qui commence par 02.

SQLEXCEPTION

Un code d'erreur SQL qui ne commence ni par 01 ni par 02.

nom_condition

Une condition d'erreur définie au préalable avec l'instruction DECLARE... CONDITION.

Les mots clés CONTINUE et EXIT permettent de définir ce qui se passe après l'exécution des instructions du gestionnaire. Avec CONTINUE, l'exécution du programme se poursuit. Avec EXIT, l'exécution du bloc BEGIN END qui contient la déclaration du gestionnaire se termine ; si le bloc est imbriqué dans un bloc parent, l'exécution peut par contre se poursuivre dans le bloc parent (il n'y a plus d'erreur, puisqu'elle a été interceptée).

Si une erreur se produit et qu'aucun gestionnaire n'a été défini pour cette erreur, l'action par défaut est EXIT.

Une erreur peut être ignorée en utilisant un bloc vide dans le gestionnaire :

```
DECLARE ... HANDLER FOR ... BEGIN END
```

Lors de l'utilisation de blocs imbriqués, les erreurs se propagent de la manière suivante :

- Si une erreur se produit dans un bloc imbriqué et que cette erreur est gérée (interceptée) dans le bloc

imbriqué, l’exécution peut se poursuivre soit dans le bloc imbriquée (gestionnaire CONTINE) soit dans le bloc parent (gestionnaire EXIT).

- Si une erreur se produit dans un bloc imbriqué et que cette erreur n’est pas gérée dans le bloc imbriqué, l’exécution du bloc imbriqué se termine (EXIT par défaut) et MySQL recherche un gestionnaire pour l’erreur dans le bloc parent. S’il n’y en a pas, l’exécution du bloc parent se termine aussi (EXIT par défaut) ; s’il y en a un, il est exécuté et l’exécution se poursuit ou non dans le bloc parent selon la nature du gestionnaire (CONTINUE ou EXIT).

*Exemple*

```
mysql> CREATE PROCEDURE ps_creer_rubrique
    ->   (
    ->   -- Titre de la nouvelle rubrique.
    ->   IN p_titre VARCHAR(20),
    ->   -- Titre de la rubrique parent.
    ->   IN p_titre_parent VARCHAR(20),
    ->   -- Identifiant de la nouvelle rubrique.
    ->   OUT p_id INT
    ->   )
    -> BEGIN
    ->   -- Identifiant de la rubrique parent.
    ->   DECLARE v_id_parent INT;
    ->   -- Indicateur d’existence de la rubrique parent.
    ->   DECLARE v_parent_existe BOOLEAN;
    ->   -- Lire l’identifiant de la rubrique parent
    ->   -- (si le titre de la rubrique parent passé en
    ->   -- paramètre est non vide).
    ->   IF p_titre_parent IS NOT NULL
    ->   THEN
    ->     -- Utiliser un bloc imbriqué avec un
    ->     -- gestionnaire d’erreur.
    ->     -- Si le parent n’est pas trouvé, positionner
    ->     -- l’indicateur à FALSE et quitter le sous-bloc.
    ->     BEGIN
    ->       DECLARE EXIT HANDLER FOR NOT FOUND
    ->       SET v_parent_existe = FALSE;
    ->       SELECT id INTO v_id_parent
    ->       FROM rubrique WHERE titre = p_titre_parent;
    ->       SET v_parent_existe = TRUE; -- c’est bon !
    ->     END;
    ->   ELSE
    ->     -- Pas de rubrique parent passé en paramètre.
    ->     -- Considérer que le parent existe.
    ->     SET v_parent_existe = TRUE;
    ->   END IF;
    ->   -- Si le parent existe
    ->   IF v_parent_existe
    ->   THEN
    ->     -- Insérer la nouvelle rubrique.
    ->     INSERT INTO rubrique (titre,id_parent)
    ->     VALUES (p_titre,v_id_parent);
    ->     -- Lire l’identifiant de la nouvelle rubrique.
    ->     SET p_id = LAST_INSERT_ID();
    ->   ELSE
    ->     -- La rubrique parent n’existe pas, retourner -1.
    ->     SET p_id = -1;
    ->   END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
mysql>
mysql> -- Créer une rubrique parent.
mysql> CALL ps_creer_rubrique(’Bureautique’,NULL,@id);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM rubrique WHERE id = @id;
+----+-------------+-----------+
| id | titre       | id_parent |
+----+-------------+-----------+
| 19 | Bureautique |      NULL |
+----+-------------+-----------+
1 row in set (0.00 sec)

mysql> -- Créer une sous-rubrique de la rubrique "Bureautique".
mysql> CALL ps_creer_rubrique('Tableur','Bureautique',@id);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM rubrique WHERE id = @id;
+----+---------+-----------+
| id | titre   | id_parent |
+----+---------+-----------+
| 20 | Tableur |        19 |
+----+---------+-----------+
1 row in set (0.00 sec)

mysql> -- Créer une sous-rubrique pour une rubrique parent
mysql> -- qui n'existe pas.
mysql> CALL ps_creer_rubrique('Suite','Burotik',@id);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @id;
+------+
| @id  |
+------+
| -1   |
+------+
1 row in set (0.00 sec)
```

### f. Les curseurs

Un curseur est une construction du langage qui permet de parcourir le résultat d'une requête SELECT.

L'utilisation d'un curseur est similaire au parcours d'un fichier séquentiel. L'utilisation d'un curseur nécessite 4 étapes :

- déclarer le curseur ;
- ouvrir le curseur ;
- lire la ligne courante dans des variables ;
- fermer le curseur.

Un curseur peut être déclaré à l'aide de l'instruction DECLARE.

*Syntaxe*

```
DECLARE nom_curseur CURSOR FOR ordre_SELECT
```

nom_curseur est le nom donné au curseur.

ordre_SELECT définit la requête SQL du curseur. Toutes les clauses habituelles d'une requête SELECT (sauf INTO) peuvent être utilisées. La requête n'est pas exécutée à ce stade.

Un programme peut utiliser plusieurs curseurs (avec des noms différents).

 Les curseurs doivent être déclarés après les variables et les conditions mais avant les gestionnaires.

L'instruction OPEN permet d'ouvrir un curseur défini au préalable.

*Syntaxe*

```
OPEN nom_curseur
```

Lors de l'ouverture du curseur, la requête est exécutée et un pointeur interne est positionné sur la première ligne du résultat ; aucun résultat n'est retourné.

L'instruction FETCH permet de lire la ligne courante du curseur dans des variables et de faire avancer le pointeur interne sur la ligne suivante.

*Syntaxe*

```
FETCH nom_curseur INTO nom_variable[,...]
```

La clause INTO doit contenir autant de variables qu'il y a d'expressions dans le SELECT du curseur.

S'il n'y a plus aucune ligne à lire, l'instruction FETCH lève l'erreur "No data" :

```
ERROR 1329 (02000): No data - zero rows fetched, selected, or processed
```

Un gestionnaire associé à la condition NOT FOUND peut être écrit pour détecter cette erreur (voir ci-après).

L'instruction CLOSE permet de fermer un curseur ouvert au préalable.

*Syntaxe*

```
CLOSE nom_curseur
```

La plupart du temps, le programme doit lire et extraire toutes les lignes du curseur. Pour faire cela, l'instruction FETCH est écrite dans une structure de contrôle itérative et un gestionnaire associé à la condition NOT FOUND est écrit pour détecter la fin du curseur et permettre une sortie de boucle propre (sans erreur).

*Exemple*

```
mysql> delimiter //
mysql>
mysql> CREATE FUNCTION fs_rubriques_livre
    ->   (
    ->   p_id_livre INT -- identifiant d'un livre
    ->   )
    ->   RETURNS TEXT
    -> BEGIN
    ->   -- Titre d'une rubrique.
    ->   DECLARE v_titre VARCHAR(20);
    ->   -- Résultat de la fonction.
    ->   DECLARE v_resultat TEXT DEFAULT '';
    ->   -- Indicateur de fin de parcours du curseur.
    ->   DECLARE v_fin BOOLEAN DEFAULT FALSE;
    ->   -- Curseur qui sélectionne les rubriques (parents)
    ->   -- d'un livre.
    ->   DECLARE cur_rubriques CURSOR FOR
    ->   SELECT
    ->     rub.titre
    ->   FROM
    ->     rubrique rub -- rubrique parent
    ->       JOIN
    ->     rubrique sru -- sous-rubrique
    ->       ON (rub.id = sru.id_parent)
    ->       JOIN
    ->     rubrique_livre rul -- (sous-)rubrique d'un livre
    ->       ON (sru.id = rul.id_rubrique)
    ->   WHERE rul.id_livre = p_id_livre;
    ->   -- Gestionnaire de détection de la fin du curseur.
    ->   DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_fin = TRUE;
    ->   -- Ouvrir le curseur.
    ->   OPEN cur_rubriques;
    ->   -- Boucle permettant de parcourir le résultat du curseur.
    ->   curseur: LOOP
    ->     -- Lire la ligne courante du curseur.
    ->     FETCH cur_rubriques INTO v_titre;
    ->     -- Quitter la boucle si tout a été lu.
```

```
    ->     IF v_fin
    ->     THEN
    ->       LEAVE curseur;
    ->     END IF;
    ->     -- Ajouter la rubrique dans la liste.
    ->     SET v_resultat = CONCAT(v_resultat,',',v_titre);
    ->   END LOOP curseur;
    ->   -- Fermer le curseur.
    ->   CLOSE cur_rubriques;
    ->   -- Retourner le résultat (enlever la virgule de tête).
    ->   RETURN TRIM(',' FROM v_resultat);
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;
mysql>
mysql> -- Test de la fonction.
mysql> SELECT titre,fs_rubriques_livre(id) rubriques
    -> FROM livre LIMIT 1 ;
+---------+-----------------------------------+
| titre   | rubriques                         |
+---------+-----------------------------------+
| PHP 5.2 | Développement,Internet,Open Source |
+---------+-----------------------------------+
1 row in set (0.00 sec)
```

### g. Récursivité

Les procédures récursives (procédure qui s'appelle elle-même) sont autorisées sous réserve de donner une valeur différente de zéro à la variable système `max_sp_recursion_depth` (maximum autorisé = 255).

Les fonctions récursives sont interdites.

# Développer des triggers

## 1. Définition

Un trigger (déclencheur en français) est un programme stocké associé à une table et qui se déclenche automatiquement lorsqu'un événement de mise à jour (INSERT, UPDATE ou DELETE) survient sur la table. Un trigger n'est jamais explicitement exécuté par un autre programme.

Les triggers permettent d'implémenter des règles de gestion côté serveur. Les principales utilisations des triggers sont les suivantes :

- Calculer automatiquement la valeur d'une colonne : par exemple, un trigger peut être utilisé pour calculer automatiquement un prix TTC à partir d'un prix HT et d'un taux de TVA.
- Auditer les mises à jour effectuées dans la base de données : par exemple, à chaque fois qu'un article est supprimé, un trigger garde la trace de la suppression (qui, quand, quoi) dans une table d'audit.

Les triggers sont supportés dans MySQL depuis la version 5.0.2.

## 2. Gestion des triggers

L'ordre SQL CREATE TRIGGER permet de créer un trigger.

*Syntaxe :*

```
CREATE TRIGGER [nom_base.]nom_trigger
  {BEFORE | AFTER}
  {INSERT | UPDATE | DELETE}
  ON nom_table
  FOR EACH ROW
BEGIN
  instructions;
END;
```

nom_base désigne la base de données dans laquelle le trigger doit être défini. Par défaut, le programme est stocké dans la base de données courante.

nom_trigger spécifie le nom du trigger. Ce nom doit respecter les règles de nommage des objets MySQL.

nom_table spécifie la table à laquelle le trigger est associé ; la table et le trigger doivent être stockés dans la même base de données.

La clause BEFORE ou AFTER permet d'indiquer à quel moment le trigger se déclenche : juste avant que la mise à jour se produise (BEFORE) ou juste après (AFTER).

Le mot clé INSERT, UPDATE ou DELETE indique sur quel ordre SQL de mise à jour le trigger doit se déclencher. Il faut noter que l'instruction LOAD DATA déclenche les triggers INSERT ; à l'inverse les instructions DROP TABLE et TRUNCATE ne déclenchent pas les triggers DELETE.

La clause FOR EACH ROW indique que le trigger se déclenche pour chaque ligne mise à jour (trigger de niveau ligne). MySQL ne supporte pas (encore ?) les triggers de niveau instruction qui ne se déclencheraient qu'une fois pour la totalité de l'ordre, indépendamment du nombre de lignes mises à jour.

Il ne peut pas y avoir deux triggers définis sur une même table avec les mêmes conditions de déclenchement.

Les instructions qui composent le code du trigger sont incluses entre les mots clés BEGIN et END. Si le code du trigger ne comporte qu'une seule instruction, les mots clés BEGIN et END peuvent être omis. Le code d'un trigger utilise les mêmes constructions et les mêmes règles de syntaxe que les programmes stockés. Les programmes stockés peuvent être appelés dans le code d'un trigger.

Dans le code du trigger, il est possible de faire référence à ligne en cours de mise à jour grâce aux identifiants prédéfinis OLD et NEW.

*Syntaxe*

```
OLD.nom_colonneNEW.nom_colonne
```

OLD.nom_colonne donne l'ancienne valeur de la colonne (avant la mise à jour) ; OLD.nom_colonne ne peut pas être utilisé dans un trigger INSERT (pas d'ancienne valeur).

NEW.nom_colonne donne la nouvelle valeur de la colonne (après la mise à jour) ; NEW.nom_colonne ne peut pas être utilisé dans un trigger DELETE (pas de nouvelle valeur).

Dans un trigger BEFORE, il est possible de modifier les nouvelles valeurs des colonnes en modifiant les identifiants NEW.nom_colonne à l'aide d'une instruction SET.

NEW.nom_colonne ne peut pas être modifié dans un trigger AFTER. Par ailleurs, OLD.nom_colonne ne peut jamais être modifié.

La lecture de NEW.nom_colonne ou OLD.nom_colonne nécessite le privilège SELECT sur la table ; la modification de NEW.nom_colonne nécessite le privilège UPDATE sur la table.

Dans un trigger BEFORE, la valeur des colonnes AUTO_INCREMENT est déterminée après l'exécution des triggers BEFORE ; pour une colonne AUTO_INCREMENT, dans un trigger BEFORE, NEW.nom_colonne vaut toujours 0.

*Exemple*

```
mysql> delimiter //

mysql> CREATE TRIGGER tr_calculer_frais_collection
    ->   BEFORE INSERT ON collection
    ->   FOR EACH ROW
    -> BEGIN
    ->   -- Calculer les frais de la collection s'ils sont
    ->   -- vides ou égaux à zéro.
    ->   IF IFNULL(NEW.frais_ht,0) = 0
    ->   THEN
    ->     -- Le montant des frais dépend du prix de vente H.T.
    ->     CASE
    ->       WHEN NEW.prix_ht <= 15
    ->       THEN
    ->         SET NEW.frais_ht = 1;
    ->       WHEN NEW.prix_ht <= 40
    ->       THEN
    ->         SET NEW.frais_ht = 1.5;
    ->       ELSE
    ->         SET NEW.frais_ht = 2;
    ->     END CASE;
    ->   END IF;
    -> END;
    -> //

mysql> delimiter ;

mysql> -- Insérer une nouvelle collection sans mentionner les frais.
mysql> INSERT INTO collection(nom,prix_ht)
    -> VALUES('Epsilon',48.63);
Query OK, 1 row affected (0.01 sec)

mysql> -- Vérifier le résultat.
mysql> SELECT * FROM collection WHERE id = LAST_INSERT_ID();
+----+---------+---------+----------+
| id | nom     | prix_ht | frais_ht |
+----+---------+---------+----------+
|  9 | Epsilon |   48.63 |     2.00 |
+----+---------+---------+----------+
1 row in set (0.01 sec)
```

Les différentes variantes de l'ordre SQL SHOW (cf. chapitre Construire une base de données dans MySQL - Obtenir des informations sur les bases de données) permettent d'afficher des informations sur les triggers.

*Syntaxe*

```
SHOW CREATE TRIGGER nom_triggerSHOW TRIGGERS [FROM nom_base] [condition]
```

Un trigger peut être supprimé à l'aide de l'ordre DROP TRIGGER.

*Syntaxe*

```
DROP TRIGGER [IF EXISTS] [nom_base.]nom_trigger
```

Lors de la suppression d'une table, les triggers associés sont supprimés.

## 3. Considérations sur l'utilisation des triggers

### a. Restrictions

Les restrictions sur les ordres SQL utilisables dans un trigger sont les mêmes que pour les fonctions :

- Instructions qui font explicitement ou implicitement un COMMIT ou un ROLLBACK ;
- Instructions qui retournent un résultat directement au client (SELECT sans INTO, SHOW, etc.) ;
- Modifier la table utilisée par l'ordre qui a déclenché le trigger ;
- Appeler une procédure qui utilise un ordre interdit dans les triggers.

De plus, l'instruction RETURN est interdite dans le code d'un trigger ; par contre, l'instruction LEAVE peut être utilisée pour quitter immédiatement un trigger.

Les triggers ne sont pas déclenchés par les suppressions en cascade des clés étrangères.

### b. Résultat en cas d'erreur

Le résultat obtenu en cas d'échec du trigger ou de l'ordre à l'origine du déclenchement du trigger dépend de la nature de la table.

#### Table non transactionnelle

Si un trigger BEFORE échoue pour une ligne, la mise à jour n'est pas exécutée sur la ligne concernée et l'instruction à l'origine du déclenchement du trigger s'arrête ; par contre, les lignes déjà mises à jour et les opérations éventuelles effectuées par le trigger sont conservées.

Si un trigger AFTER échoue pour une ligne, la mise à jour (déjà) effectuée sur la ligne concernée n'est pas annulée et l'instruction à l'origine du déclenchement du trigger s'arrête ; par contre, les lignes déjà mises à jour et les opérations éventuelles effectuées par le trigger sont conservées.

Si la mise à jour échoue pour une ligne, les triggers AFTER ne sont pas exécutés pour la ligne en question ; par contre, les triggers BEFORE ont déjà étés exécutés pour la ligne en question. Dans les deux cas, les lignes déjà mises à jour et les opérations éventuelles effectuées par les triggers sont conservées.

#### Table transactionnelle

Pour une table transactionnelle, c'est plus simple : en cas d'erreur (dans un trigger ou dans la mise à jour d'une ligne), tout ce qui a été effectué depuis le début de l'ordre est annulé.

# Qu'est-ce que PHP ?

PHP est un langage de script qui s'exécute côté serveur, le code PHP étant inclus dans une page HTML classique. Il peut donc être comparé à d'autres langages de script qui fonctionnent sur le même principe : ASP (*Active Server Pages*) ou JSP (*Java Server Pages*).

À la différence d'un langage comme le JavaScript, où le code est exécuté côté client (dans le navigateur), le code PHP est exécuté côté serveur. Le résultat de cette exécution est intégré dans la page HTML qui est envoyée au navigateur. Ce dernier n'a aucune connaissance de l'existence du traitement qui s'est déroulé sur le serveur.

Cette technique permet de réaliser des pages Web dynamiques dont le contenu peut être complètement ou partiellement généré au moment de l'appel de la page, grâce à des informations récupérées dans un formulaire ou extraites d'une base de données.

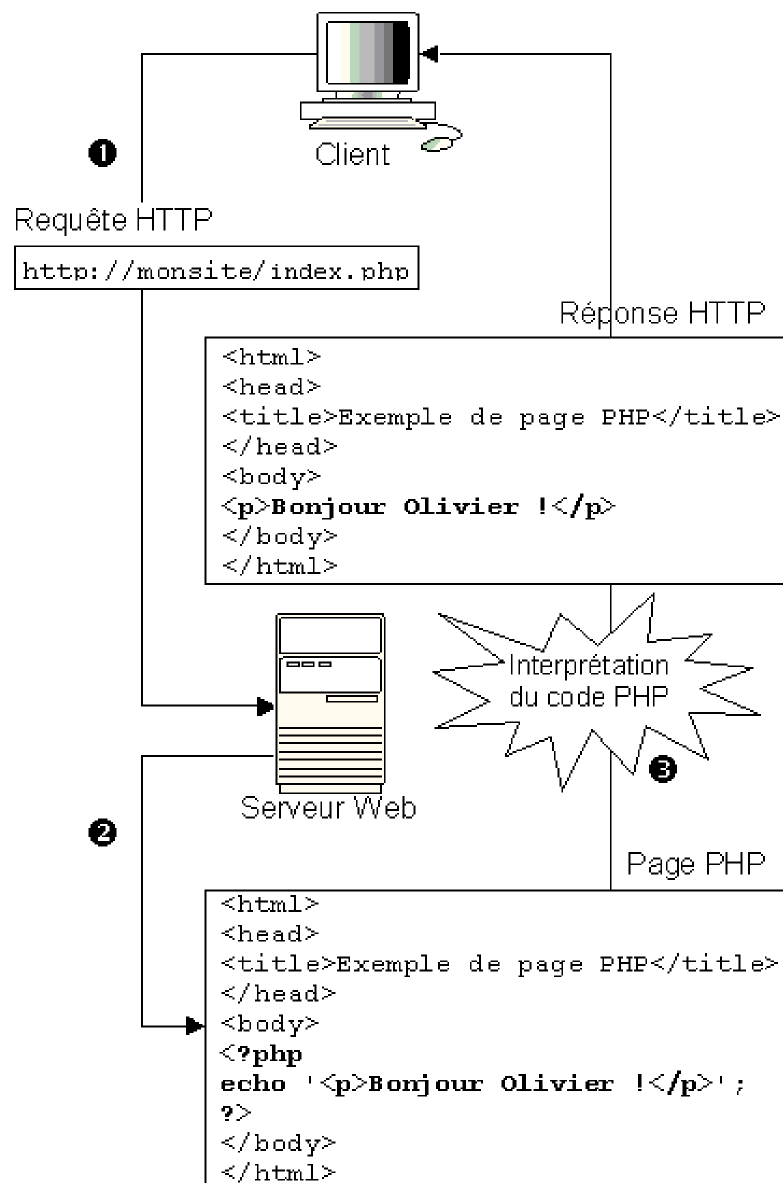
*Exemple simple de page PHP :*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
  </head>
  <body>
    <?php
     echo '<p>Bonjour Olivier !</p>';
     ?>
  </body>
</html>
```

La partie en gras est du code PHP inclus dans la page HTML à l'intérieur des balises `<?php` et `?>`. Sur cet exemple simple, le code PHP se contente d'afficher un texte statique « Bonjour Olivier ! » grâce à la fonction `echo` ; dans un vrai programme PHP, il est probable que ce texte serait généré dynamiquement en fonction de l'identification de l'utilisateur.

Pour indiquer au serveur Web qu'une page HTML contient du code PHP à exécuter, il suffit de donner au fichier une extension particulière : `.php` (sauf configuration particulière du serveur).

Le schéma suivant explique comment est traité un fichier PHP par le serveur Web.




Lorsqu'un fichier PHP est demandé au serveur Web, le code PHP inclus dans la page HTML est d'abord exécuté sur le serveur. Le résultat de cette exécution est inséré dans la page à la place du code PHP et la page est renvoyée au navigateur.

D'autres langages, comme le PERL (*Practical Extraction and Report Language*) ou le C, permettent d'écrire des scripts CGI (*Common Gateway Interface*) dans le but d'obtenir le même résultat. Ces langages sont souvent considérés comme moins pratiques et moins lisibles que PHP pour réaliser une page Web dynamique. En effet, le programme doit générer l'intégralité de la page HTML alors qu'en PHP, seule la partie réellement dynamique sera codée à l'intérieur de la page.

De plus, PHP a été écrit spécialement pour le Web et possède des fonctionnalités parfaitement adaptées à ce type de développement, ce qui n'est le cas ni de PERL ni du C (qui sont par ailleurs d'excellents langages).

# Structure de base d'une page PHP

## 1. Les balises PHP

Comme nous l'avons vu précédemment, le code PHP est inclus dans une page HTML à l'intérieur de balises (aussi appelées *tags*).

PHP accepte quatre syntaxes pour les balises :

- `<?php ... ?>`
- `<script language="php"> ... </script>`
- `<? ... ?>`
- `<% ... %>`

La première syntaxe est la syntaxe habituelle, recommandée.

La deuxième syntaxe, plus lourde, utilise la balise standard `script` ; elle peut être utile si votre éditeur HTML interprète mal les autres syntaxes.

La troisième syntaxe n'est envisageable que si elle a été autorisée dans le fichier de paramétrage de PHP (`php.ini`) en mettant le paramètre `short_open_tag` à `on`.

La quatrième syntaxe permet d'employer la balise ASP mais elle est utilisable uniquement si elle a été autorisée dans le fichier de paramétrage de PHP en mettant le paramètre `asp_tags` à `on`.

## 2. La fonction echo

La fonction `echo` est la fonction de base de toute page PHP. Elle permet d'afficher une ou plusieurs chaînes et donc d'inclure du texte dans la page HTML envoyée au navigateur.

*Syntaxe* :

```
echo(chaîne texte)
echo chaîne texte[,...]
```

`texte`

Texte à afficher.

La première syntaxe n'accepte qu'un paramètre alors que la deuxième en accepte plusieurs.

*Exemple :*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
  </head>
  <body>
    <p>
    <?php
    echo('Bonjour Olivier !');
    ?>
    <br />
    <?php
    echo 'Bonjour ','Valérie ','!';
    ?>
    </p>
  </body>
</html>
```

*Résultat :*

```
Bonjour Olivier !
Bonjour Valérie !
```

Il n'y pas de saut de ligne automatique dans le résultat de l'exécution du code PHP. En cas de besoin, il est donc nécessaire d'insérer la balise HTML `<br />` qui provoque un saut de ligne dans la page HTML finale (voir l'exemple ci-dessus).

Le texte passé en paramètre à la fonction `echo` peut être écrit sur plusieurs lignes dans le source mais il est affiché sur une seule dans le résultat.

## 3. Séparateur d'instruction

En PHP, toutes les instructions doivent se terminer par un point-virgule.

*Exemple :*

```
<?php
echo 'Bonjour ';
echo 'Olivier !';
?>
```

*Résultat :*

```
Bonjour Olivier !
```

En cas d'omission, une erreur est générée. La seule exception concerne l'instruction qui précède la balise de fin pour laquelle le point-virgule peut être omis.

Plusieurs instructions peuvent être écrites sur la même ligne du moment qu'elles sont séparées par un point-virgule. Néanmoins, cette écriture nuit parfois à la lisibilité du code.

*Exemple* :

```
<?php
echo 'Bonjour '; echo 'Olivier !';
?>
```

## 4. Commentaire

PHP propose deux syntaxes :

- `//` ou `#` pour insérer du commentaire jusqu'à la fin de la ligne
- `/* ... */` pour insérer du commentaire sur plusieurs lignes

*Exemple :*

```
<?php
// commentaire sur une seule ligne
# commentaire sur une seule ligne
/* commentaire sur
plusieur lignes */
echo 'Bonjour '; // commentaire jusqu'à la fin de la ligne
echo 'Olivier !'; # commentaire jusqu'à la fin de la ligne
?>
```

 Les commentaires `/* ... */` ne doivent pas être imbriqués.

## 5. Mixer du PHP et de l'HTML

Il existe de nombreuses approches pour mixer du PHP et de l'HTML.

Ces différentes approches reposent néanmoins sur deux principes très simples :

- La page peut contenir une ou plusieurs inclusion(s) de code PHP.

- Le code PHP génère du « texte » qui est intégré dans la page HTML envoyée au navigateur. Tout « texte » compréhensible par le navigateur peut donc être généré par le code PHP : du texte simple, du code HTML, du code JavaScript, ...

Les exemples qui suivent utilisent des variables et des fonctions PHP (récupération de la date et de l'heure). Ces notions seront présentées plus en détail dans la suite de cet ouvrage.

*Exemple de page contenant du code PHP en plusieurs endroits :*

```
<?php
// Déclaration de variables qui seront utilisées plus loin.
// Cette section de code PHP ne génère par de sortie dans la
// page HTML (pas d'appel à echo).
$nom = 'Olivier'; // nom de l'utilisateur
$titre_page = 'Les éditions ENI présentent ...'; // titre de la page
$aujourdhui = date("d/m/Y"); // date du jour
$heure = date("H:i:s"); // heure
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
    <?php /* affichage du titre */ echo $titre_page; ?>
    </title>
  </head>
  <body>
    <p>
    <?php
     /* Affichage du nom de l'utilisateur.
     ** Les tags de mise en gras du nom (<b>) et de retour à
     ** la ligne (<br />) sont inclus dans la chaîne envoyée
     ** par le echo.
     */
     echo "Bonjour <b>$nom</b> !<br />";
     // Affichage de la date et de l'heure.
     echo "Nous sommes le $aujourdhui ; il est $heure.";
     ?>
    </p>
  </body>
</html>
```

*Résultat :*

```
Bonjour Olivier !
Nous sommes le 27/01/2008 ; il est 10:48:52.
```

*Source de la page dans le navigateur (les éléments générés par PHP sont en gras) :*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>
    Les éditions ENI présentent ...    </title>
  </head>
  <body>
    <p>
    Bonjour <b>Olivier</b> !<br />Nous sommes le 27/01/2008 ; il est 10:48:52.    </p>
```

```
  </body>
</html>
```

*Exemple de page générée entièrement par du code PHP (suivant le principe CGI) :*

```
<?php
// Déclaration de variables qui sont utilisées plus loin.
$nom = 'Olivier'; // nom de l'utilisateur
$titre_page = 'Les éditions ENI présentent ...'; // titre de la page
$aujourdhui = date("d/m/Y"); // date du jour
$heure = date("H:i:s"); // heure
// Génération des balises d'ouverture du document HTML.
echo '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" ',
     '"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">';
echo '<html xmlns="http://www.w3.org/1999/xhtml">';
echo '<head>';
echo "<title>$titre_page</title>";
echo '</head>';
echo '<body>';
echo '<p>';
 /* Affichage du nom de l'utilisateur.
 ** Les tags de mise en gras du nom (<b>) et de retour à la ligne
 ** (<br />) sont inclus dans la chaîne envoyée par le echo.
 */
echo "Bonjour <b>$nom</b> !<br />";
// Affichage de la date et de l'heure.
echo "Nous sommes le $aujourdhui ; il est $heure.";
echo '</p>';
echo '</body>';
echo '</html>';
?>
```

*Résultat :*

Bonjour **Olivier** !
Nous sommes le 27/01/2008 ; il est 11:03:27.

Source de la page dans le navigateur (tout est sur une ligne) :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"><html
xmlns="http://www.w3.org/1999/xhtml"><head><title>Les
éditions ENI présentent ...</title></head><body><p>Bonjour
<b>Olivier</b> !<br />Nous sommes le 27/01/2008 ; il est
11:03:27.</p></body></html>
```

Il n'y a pas de règle pour mixer du PHP et de l'HTML. Une approche couramment employée par les développeurs consiste à utiliser PHP uniquement pour générer la partie réellement dynamique de la page ; le reste est directement écrit en HTML dans le fichier. Cette technique rend le code moins lourd et permet de voir tout de suite où se trouve la logique applicative.

Le document HTML envoyé au navigateur doit être valide et si possible conforme aux standards HTML ou XHTML du W3C (*World Wide Web Consortium*). Si besoin, n'hésitez pas à utiliser le service de validation du W3C (http://validator.w3.org/).

Dans cet ouvrage, les exemples respectent au maximum la recommandation XHTML 1.0. Néanmoins, pour des raisons de place, les exemples n'intègrent pas systématiquement les déclarations initiales :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
...
```

## 6. Règles de nommage

Toute entité PHP nommée (variable, constante, fonction, ...) doit avoir un nom qui respecte les règles suivantes :

- commencer par une lettre ou un souligné (_) ;
- comporter ensuite des lettres, des chiffres ou le caractère souligné.

Dans cette définition, une lettre représente toute lettre minuscule ou majuscule comprise entre a et z (a à z et A à Z) ainsi que tout caractère de code ASCII compris entre 127 et 255. Les caractères accentués sont donc autorisés, mais pas les caractères du type #$%& qui ont une signification spéciale dans le langage PHP.

# Configuration de PHP

## 1. Le fichier de configuration php.ini

Tout au long de cet ouvrage, nous rencontrerons plusieurs directives de configuration qui peuvent être utilisées pour modifier le comportement de PHP.

Ces directives de configuration sont saisies dans le fichier de paramétrage de PHP (`php.ini`).

PHP fournit deux exemples de fichier `php.ini` : `php.ini-dist` et `php.ini-recommended`.

Le fichier `php.ini-dist` est un exemple de fichier de configuration, plutôt destiné à être utilisé dans un environnement de développement. À l'inverse, le fichier `php.ini-recommended` est plutôt destiné à être employé dans un environnement d'exploitation ; il contient des réglages qui rendent PHP plus sécurisé et/ou performant.

Ces deux fichiers comportent beaucoup de commentaires qui expliquent le rôle de chaque directive et donnent des conseils sur leur usage.

Pour utiliser un de ces fichiers, copiez-le à l'emplacement approprié sur votre plate-forme et renommez-le en `php.ini`. Le fichier `php.ini` est notamment recherché dans les endroits suivants (dans cet ordre) :

- un endroit spécifique au serveur Web (par exemple, la directive `PHPIniDir` d'Apache 2) ;
- un endroit défini par la variable d'environnement `PHPRC` ;
- le dossier `/usr/local/lib` sous Linux/Unix et `c:\windows` ou `c:\winnt` sous Windows.

Dans cet ouvrage, et sauf indication contraire, nous supposerons que deux directives relatives à la gestion des erreurs sont positionnées de la manière suivante :

```
<$I[]display_errors>display_errors = on
```

Les erreurs sont affichées.

```
<$I[]error_reporting>error_reporting = E_ALL & ~E_NOTICE
```

Toutes les erreurs sont affichées, sauf les erreurs de niveau `E_NOTICE` (simples informations, par exemple lors de l'utilisation d'une variable non initialisée).

La gestion des erreurs est présentée en détail dans le chapitre Gérer les erreurs dans un script PHP.

## 2. Informations sur la configuration

PHP propose deux fonctions particulièrement utiles pour obtenir des informations sur la configuration : `phpversion` et `phpinfo`.

La fonction `phpversion` retourne le numéro de version de PHP et la fonction `phpinfo` affiche une grande quantité d'informations sur la configuration de PHP et son environnement. Le numéro de version est aussi disponible via la constante prédéfinie `PHP_VERSION`.

*Syntaxe*

```
chaîne phpversion(chaîne extension)
entier phpinfo([entier quoi])
```

*Avec*

`extension`

Si ce paramètre est spécifié, la fonction retourne la version de l'extension portant ce nom (ou `FALSE` si la version est inconnue ou si l'extension n'est pas disponible).

`quoi`

Nature de l'information désirée. Utilisez une ou plusieurs (somme) des constantes suivantes : INFO_GENERAL (1) : informations générales (version, emplacement du fichier php.ini, système d'exploitation, etc.). INFO_CREDITS (2) : informations sur les auteurs. INFO_CONFIGURATION (4) : informations sur la configuration (valeurs des directives). INFO_MODULES (8) : informations sur les modules chargés, avec leur configuration respective. INFO_ENVIRONMENT (16) : informations sur l'environnement (voir la variable $_ENV en annexe). INFO_VARIABLES (32) : valeurs de toutes les variables prédéfinies (voir l'annexe). INFO_LICENSE (64) : informations sur la licence. INFO_ALL (-1) : toutes les informations (valeur par défaut).

phpinfo retourne TRUE en cas de succès et FALSE en cas d'erreur.

*Exemple 1 :*

```
<?php
echo phpversion();
?>
```

*Résultat :*

```
5.2.4
```

*Exemple 2 :*

```
<?php
// informations générales et informations
// de licence
phpinfo(INFO_GENERAL+INFO_LICENSE);
?>
```

*Résultat :*

# PHP Version 5.2.4



| | |
|---|---|
| **System** | Linux xampp 2.6.9-42.0.3.ELsmp #1 SMP Fri Oct 6 06:21:39 CDT 2006 i686 |
| **Build Date** | Sep 30 2007 09:09:10 |
| **Configure Command** | ./configure --prefix=/opt/lampp --with-apxs2=/opt/lampp/bin/apxs --with-config-file-path=/opt/lampp/etc --with-mysql=/opt/lampp --enable-inline-optimization --disable-debug --enable-bcmath --enable-calendar --enable-ctype --enable-dbase --enable-discard-path --enable-exif --enable-filepro --enable-force-cgi-redirect --enable-ftp --enable-gd-imgstrttf --enable-gd-native-ttf --with-ttf --enable-magic-quotes --enable-memory-limit --enable-shmop --enable-sigchild --enable-sysvsem --enable-sysvshm --enable-track-vars --enable-trans-sid --enable-wddx --enable-yp --with-ftp --with-gdbm=/opt/lampp --with-jpeg-dir=/opt/lampp --with-png-dir=/opt/lampp --with-freetype-dir=/opt/lampp --without-xpm --with-zlib=yes --with-zlib-dir=/opt/lampp --with-openssl=/opt/lampp --with-expat-dir=/opt/lampp --enable-xslt=/opt/lampp --with-xsl=/opt/lampp --with-dom=/opt/lampp --with-ldap=/opt/lampp --with-ncurses=/opt/lampp --with-gd --with-imap-dir=/opt/lampp --with-imap-ssl --with-imap=/opt/lampp --with-gettext=/opt/lampp --with-mssql=/opt/lampp --with-sybase=/opt/lampp --with-interbase=shared,/opt/interbase --with-mysql-sock=/opt/lampp/var/mysql/mysql.sock --with-oci8=shared,instantclient,/opt/lampp/lib/instantclient --with-mcrypt=/opt/lampp --with-mhash=/opt/lampp --enable-sockets --enable-mbstring=all --with-curl=/opt/lampp --enable-mbregex --enable-zend-multibyte --enable-exif --with-bz2=/opt/lampp --with-sqlite=shared,/opt/lampp --with-libxml-dir=/opt/lampp --enable-soap --enable-pcntl --with-mysqli=/opt/lampp/bin/mysql_config --with-mime-magic --with-pgsql=shared,/opt/lampp/postgresql --with-iconv --enable-dio --with-pdo-mysql=/opt/lampp --with-pdo-pgsql=/opt/lampp/postgresql --with-pdo-sqlite --with-ming=shared,/opt/lampp |
| **Server API** | Apache 2.0 Handler |
| **Virtual Directory Support** | disabled |
| **Configuration File (php.ini) Path** | /opt/lampp/etc |
| **Loaded Configuration File** | /opt/lampp/etc/php.ini |
| **PHP API** | 20041225 |
| **PHP Extension** | 20060613 |
| **Zend Extension** | 220060519 |
| **Debug Build** | no |
| **Thread Safety** | disabled |
| **Zend Memory Manager** | enabled |
| **IPv6 Support** | enabled |
| **Registered PHP Streams** | php, file, data, http, ftp, compress.bzip2, compress.zlib, https, ftps, zip |
| **Registered Stream Socket Transports** | tcp, udp, unix, udg, ssl, sslv3, sslv2, tls |
| **Registered Stream Filters** | string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, convert.iconv.*, bzip2.*, zlib.* |

This program makes use of the Zend Scripting Language Engine: Powered By

Zend Engine v2.2.0, Copyright (c) 1998-2007 Zend Technologies




## PHP License

This program is free software; you can redistribute it and/or modify it under the terms of the PHP License as published by the PHP Group and included in the distribution in the file: LICENSE

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

If you did not receive a copy of the PHP license, or have any questions about PHP licensing, please contact license@php.net.

*Exemple 3 :*

```
<?php
// toutes les informations
phpinfo();
?>
```

Voir aussi les fonctions `ini_get_all`, `ini_get` et `get_loaded_extensions` qui permettent d'obtenir des informations sur les directives de compilation et les extensions chargées.

# Les bases du langage PHP

## 1. Constantes

### a. Définition

La fonction `define` permet de définir une constante.

*Syntaxe*

```
booléen define(chaîne nom, mixte valeur[, booléen sensible_casse)
```

`nom`

Nom de la constante (cf. dans ce chapitre - Structure de base d'une page PHP - Règles de nommage).

`valeur`

Valeur de la constante.

`sensible_casse`

Indique si le nom de la constante est sensible à la casse (`TRUE` - valeur par défaut) ou non (`FALSE`).

La fonction `define` retourne `TRUE` en cas de succès et `FALSE` en cas d'erreur.

Tout type de données scalaire (cf. dans ce chapitre - Les bases du langage PHP - Types de données) peut être utilisé comme type de données d'une constante.

Le nom d'une constante ne doit pas commencer par un $ car ce préfixe est réservé au nom des variables (cf. dans ce chapitre - Les bases du langage PHP - Variables). Il faut noter que définir une constante dont le nom commence pas un $ ne génère pas d'erreur (`define` retourne `TRUE` !), mais, à l'utilisation, la constante sera vue comme une variable non initialisée.

Une fois définie, une constante n'est plus modifiable par la suite, ni par un nouvel appel à `define` (retourne `FALSE` et laisse la valeur de la constante inchangée) ni par une affectation directe (génère une erreur d'analyse du script).

*Exemples*

```
<?php
// Définir une constante (dont le nom est par défaut
// sensible à la casse).
define('CONSTANTE','valeur de CONSTANTE');
// Afficher la valeur de CONSTANTE (=> OK).
echo 'CONSTANTE = ',CONSTANTE,'<br />';
// Afficher la valeur de constante (=> vide)
echo 'constante = ',constante;
echo ' => interprété en littéral<br />';
?>
```

*Résultat*

```
CONSTANTE = valeur de CONSTANTE
constante = constante => interprété en littéral
```

Traditionnellement, les noms des constantes sont définis en majuscules.

Utiliser une constante non définie (ou une variable non initialisée) ou tenter de redéfinir une constante déjà définie génèrent une erreur de niveau `E_NOTICE`. Le niveau d'erreur effectivement signalé par PHP dépend de directives de configuration dans le fichier `php.ini` (cf. chapitre Gérer les erreurs dans un script PHP). Le résultat précédent correspond à une configuration dans laquelle les erreurs de niveau `E_NOTICE` ne sont pas affichées.

### b. Portée

La portée d'une constante est le script dans lequel elle est définie. Une constante peut donc être définie dans une première section de code PHP et utilisée dans une autre section de code PHP du même script.

*Exemple*

```
<?php
// Définir une constante.
define('NOM','Olivier');
?>
```

```
<html>
<body>
<p>Bonjour <b><?php echo NOM; ?></b> !</p>
</body>
</html>
```

*Résultat*

```
Bonjour Olivier !
```

## 2. Variables

Une variable est une zone mémoire identifiée par un nom qui contient une valeur lisible ou modifiable dans le programme.

### a. Initialisation et affectation

En PHP les variables sont identifiées par le préfixe $ suivi d'un nom qui respecte les règles de nommage présentées précédemment (cf. dans ce chapitre - Structure de base d'une page PHP - Règles de nommage).

Le nom des variables est sensible à la casse : `$nom` et `$Nom` sont vues par PHP comme deux variables différentes. Ce comportement est dangereux car, en cas d'utilisation d'un mauvais nom, une nouvelle variable vide est créée avec une simple erreur de niveau `E_NOTICE` qui n'est pas forcément affichée (cf. chapitre Gérer les erreurs dans un script PHP). Il est donc primordial d'adopter une convention de nommage et de la respecter. Quelques suggestions :

- tout en minuscule (`$nom`) ;
- première lettre en majuscule et le reste en minuscule (`$Nom`) ;
- première lettre de chaque mot en majuscule et le reste en minuscule (`$NomDeFamille`).

Les variables PHP sont automatiquement définies lors de leur première utilisation. Il n'y a pas d'instruction spécifique pour créer une variable.

De plus, les variables PHP sont typées automatiquement ; lors de chaque affectation d'une valeur à une variable, le type de la variable est automatiquement défini ou redéfini (cf. dans ce chapitre - Les bases du langage PHP - Types de données).

Une valeur peut être affectée à une variable grâce à l'opérateur d'affectation = (cf. dans ce chapitre - Les bases du langage PHP - Opérateurs).

*Exemples*

```
<?php
// Initialiser une variable $nom.
$nom = 'Olivier';
// Afficher la variable $nom.
echo '$nom = ',$nom,'<br />';
// Afficher la variable $Nom.
echo '$<b>N</b>om = ',$Nom;
echo ' => vide (c\'est une autre variable)<br />';
// Modifier la valeur (et le type) de la variable $nom.
$nom = 123;
// Afficher la variable $nom.
echo '$nom = ',$nom,'<br />';
?>
```

*Résultat (les erreurs de niveau `E_NOTICE` ne sont pas affichées)*

```
$nom = Olivier
$Nom = => vide (c'est une autre variable)
$nom = 123
```

Tout au long de cet ouvrage, nous aurons l'occasion de rencontrer des variables automatiquement définies par PHP et contenant des valeurs relatives à l'environnement, à PHP, aux formulaires, aux cookies, etc.

### b. Portée et durée de vie

La portée d'une variable est le script dans lequel elle est définie. Une variable peut donc être définie dans une première section de code PHP et utilisée dans une autre section de code PHP du même script.

La durée de vie d'une variable est le temps de l'exécution du script. Lorsque le script se termine, les variables sont supprimées. Si le

même script est appelé plus tard, de nouvelles variables sont définies.

*Exemple*

```
<?php
// Afficher le contenu de la variable $nom.
echo '$nom = ',$nom,'<br />';
// Initialiser la variable $nom.
$nom = 'Olivier';
// Afficher de nouveau le contenu de la variable $nom.
echo '$nom = ',$nom,'<br />';
?>
```

Résultat du premier appel du script

```
$nom =
$nom = Olivier
```

Résultat du deuxième appel du script

```
$nom =
$nom = Olivier
```

Entre les deux appels, la variable a été supprimée. Au début du deuxième appel, elle ne contient plus la valeur qu'elle avait à la fin du premier appel (ce n'est plus la même variable).

---

Nous verrons dans les chapitre Gérer les liens et les formulaires avec PHP et Gérer les sessions comment conserver la valeur d'une variable au-delà de l'exécution du script ou comment transmettre la valeur d'une variable d'un script à un autre.

---

### c. Variable dynamique (ou variable variable)

PHP propose une fonctionnalité de variable dynamique (aussi appelée variable `variable`) utile dans certaines situations.

Le principe consiste à un utiliser une variable qui stocke le nom d'une autre variable et d'utiliser ensuite une notation du type `$$variable` ou `${$variable}`. Avec cette notation le `$variable` "intérieur" est remplacé par la valeur de la variable `$variable` (`valeur` par exemple) qui est alors utilisé comme nom de variable par le `$` "extérieur" (soit `$valeur` sur notre exemple).

*Exemple*

```
<?php
$une_variable = 10;
$nom_variable = 'une_variable';
echo '$une_variable = ',$une_variable,'<br />';
echo '$nom_variable = ',$nom_variable,'<br />';
echo '$$nom_variable = ',$$nom_variable,'<br />';
?>
```

*Résultat*

```
$une_variable = 10
$nom_variable = une_variable
$$nom_variable = 10
```

## 3. Types de données

### a. Types disponibles

PHP propose quatre types de données scalaires (ne pouvant contenir qu'une valeur), deux types composés (pouvant contenir plusieurs valeurs) et deux types spéciaux :

- Types scalaires :
  - nombre entier ;
  - nombre à virgule flottante ;
  - chaîne de caractères ;

- booléen.

- Types composés :

  - tableau (cf. 4 - Tableaux) ;

  - objet (cf. chapitre 8).

- Types spéciaux :

  - NULL ;

  - ressource.

## b. Types de données scalaires

### Entier

Le type entier (*integer*) permet de stocker un nombre entier signé sur 32 bits, soit des valeurs comprises entre -2 147 483 648 (-2^31) et +2 147 483 647 (+2^31-1).

En cas de dépassement de capacité dans un calcul, le résultat est automatiquement converti en nombre à virgule flottante.

### Nombre à virgule flottante

Le type nombre à virgule flottante (*float*) permet de stocker un nombre décimal sur une plage de valeurs dépendante de la plate-forme (généralement de l'ordre de $10^{-308}$ à $10^{+308}$).

Un tel nombre peut être exprimé en notation décimale `x.y` (par exemple `123.456`) ou en notation scientifique `x.yEz` ou `x.yez` (par exemple `1.23456E2`).

En cas de conversion d'un nombre à virgule flottante en entier, le nombre est tronqué (pas arrondi) à l'entier inférieur (1.9 donne 1 par exemple). En cas de dépassement de capacité, aucun message n'est affiché, mais la valeur à l'arrivé est indéfinie.

---

Des librairies particulières (aussi appelées bibliothèques) sont proposées par PHP pour traiter les nombres de grande taille (librairies BC ou GMP).

---

### Chaîne de caractères

Le type chaîne de caractères (*string*) permet de stocker toute séquence de caractères sur un octet (code ASCII compris entre 0 et 255), sans limite de taille.

Une expression littérale de type chaîne de caractères peut être spécifiée entre guillemets (`"ceci est une chaîne"`) ou entre apostrophes (`'ceci aussi est une chaîne'`) avec des différences de comportement très importantes qui sont exposées ci-après.

Les guillemets présents dans une chaîne délimitée par des guillemets ou les apostrophes présents dans une chaîne délimitée par des apostrophes doivent être "échappés", c'est-à-dire précédés du caractère anti-slash (\). En complément, un anti-slash présent en fin de chaîne, juste avant le guillemet ou l'apostrophe final, doit lui aussi être échappé par un anti-slash.

*Exemple*

```
<?php
echo 'C\'est l\'été.<br />';
echo "Je dis \"bonjour\".<br />";
?>
```

*Résultat*

```
C'est l'été.
Je dis "bonjour".
```

Une chaîne peut être saisie sur plusieurs lignes mais celle-ci est affichée sur une seule ligne dans le navigateur. Pour afficher une chaîne sur plusieurs lignes dans le navigateur, il faut insérer une balise `<br />`.

```
<?php
$chaîne = '<p>Je m\'appelle Olivier
et j\'habite en France.</p>';
echo $chaîne;
$chaîne = '<p>Je m\'appelle Olivier<br />
et j\'habite en France.</p>';
echo $chaîne;
?>
```

*Résultat*

```
Je m'appelle Olivier et j'habite en France.

Je m'appelle Olivier
et j'habite en France.
```

Lorsqu'une chaîne est délimitée par des guillemets, toute séquence de caractères commençant par le signe $ est interprété comme une variable et remplacée par la valeur de la variable : c'est le mécanisme de substitution des variables par leur valeur. Cette fonctionnalité, très pratique, ne marche pas avec les chaînes délimitées par des apostrophes (première différence entre les deux types de chaîne). Pour annuler ce comportement, il suffit d'échapper le signe $ avec l'anti-slash (\) pour qu'il se comporte comme un `$`.

*Exemple*

```
<?php
$nom = 'Olivier';
echo "Je m'appelle $nom.<br />";
echo 'Je m\'appelle $nom.<br />'; // ne marche pas
echo "\$nom = $nom"; // échappement du $
?>
```

*Résultat*

```
Je m'appelle Olivier.
Je m'appelle $nom.
$nom = Olivier
```

Si le nom de la variable est immédiatement suivi par d'autres caractères, il faut utiliser la syntaxe `{$variable}` ou `${variable}` pour que la substitution s'effectue correctement. Avec cette syntaxe, pour que l'accolade soit conservée, il faut la doubler (`{{$variable}}`).

*Exemple*

```
<?php
$fruit = 'pomme';
echo "Une $fruit ne coûte pas cher.<br />";
echo "Deux $fruits coûtent deux fois plus cher.<br />"; // !
echo "Deux {$fruit}s coûtent deux fois plus cher.<br />";
echo "{\$fruit} = {{$fruit}}.<br />";
?>
```

*Résultat*

```
Une pomme ne coûte pas cher.
Deux coûtent deux fois plus cher.
Deux pommes coûtent deux fois plus cher.
{$fruit} = {pomme}.
```

Il n'y a pas de mécanisme de substitution équivalent pour les constantes ; c'est une raison valable pour utiliser des variables en lieu et place de vraies constantes.

En complément, d'autres séquences d'échappement peuvent être utilisées dans les chaînes délimitées par des guillemets, mais pas dans celles délimitées par des apostrophes (deuxième différence entre les deux types de chaîne).

| **Séquence** | **Valeur** |
|---|---|
| `\n` | Saut de ligne (= LF = code ASCII 10) |
| `\r` | Retour chariot (= CR = code ASCII 13) |
| `\t` | Tabulation (= HT = code ASCII 9) |
| `\\` | \ (déjà abordé) |
| `\$` | $ (déjà abordé) |
| `\nnn` | Le caractère désigné par le code ASCII `nnn` exprimé en octal |
| `\xnn` | Le caractère désigné par le code ASCII `nn` exprimé en hexadécimal |

*Exemple*

```
<?php
echo "Je m'appelle Olivier.<br />\n";
echo "Je m'appelle \117\154\151\166\151\145\162.";
?>
```

*Résultat*

```
Je m'appelle Olivier.
Je m'appelle Olivier.
```

Rappel : un saut de ligne dans le source de la page envoyée au navigateur ne provoque pas de saut de ligne dans la page affichée. C'est le cas de la séquence "\n" utilisée dans notre exemple. Ici, la balise `<br />` provoque le retour à la ligne dans la page affichée.

Il est possible d'accéder au nième caractère d'une chaîne grâce à la notation `$x[i]`, `$x` désignant la variable de type chaîne et `i` le numéro du caractère (le premier caractère portant le numéro `0`). Les accolades peuvent aussi être utilisées, mais cette syntaxe deviendra obsolète dans une prochaine version.

*Exemple*

```
<?php
$nom = 'Olivier';
echo $nom[0],$nom{6};
?>
```

*Résultat*

```
Or
```

PHP est capable de convertir une chaîne en nombre (entier ou décimal) à l'aide des règles suivantes :

- Si le premier caractère non "blanc" (autre que espace, tabulation, LF, CR) n'est ni un chiffre, ni un point, ni le signe "moins", la chaîne est évaluée à 0 (entier).

- Dans le cas contraire, PHP extrait tous les caractères non "blancs" du début de chaîne jusqu'à rencontrer un caractère non « numérique » (non compris entre 1 et 9, différent du point, du signe "moins" et du symbole scientifique "e" ou "E") ; la séquence ainsi obtenue est convertie en entier (pas de point ni de symbole scientifique) ou en décimal (en cas de présence d'un point ou du symbole scientifique).

*Exemple*

```
<?php
echo '1 + "1" = ',(1 + "1"),'<br />';
echo '1 + "1.5" = ',(1 + "1.5"),'<br />';
echo '1 + "1.5E2" = ',(1 + "1.5E2"),'<br />';
echo '1 + "1e3" = ',(1 + "1e3"),'<br />';
echo '1 + 1abc = ',(1 + "1abc"),'<br />';
echo '1 + "1.5abcd" = ',(1 + "1.5abcd"),'<br />';
echo '1 + "1.5 abcd" = ',(1 + "1.5 abcd"),'<br />';
echo '1 + ".5" = ',(1 + ".5"),'<br />';
echo '1 + "-5" = ',(1 + "-5"),'<br />';
echo '1 + " \t\n\r 5" = ',(1 + " \t\n\r 5"),'<br />';
echo '1 + "abc1" = ',(1 + "abc1"),'<br />';
?>
```

*Résultat*

```
1 + "1" = 2
1 + "1.5" = 2.5
1 + "1.5E2" = 151
1 + "1e3" = 1001
1 + 1abc = 2
1 + "1.5abcd" = 2.5
1 + "1.5 abcd" = 2.5
1 + ".5" = 1.5
1 + "-5" = -4
1 + " \t\n\r 5" = 6
1 + "abc1" = 1
```

Le dernier exemple montre qu'une chaîne qui ne commence pas par un caractère numérique est convertie en entier égal à 0.

#### Booléen

Le type booléen (*boolean*) comporte deux valeurs : TRUE (ou true) et FALSE (ou false).

Ce type de données est principalement utilisé dans les structures de contrôle pour tester une condition (cf. dans ce chapitre - Les bases du langage PHP - Structures de contrôle).

PHP est capable de convertir tout type de données en booléen selon les règles suivantes :

| **Valeur** | **Converti en** |
|---|---|
| nombre entier 0<br>nombre décimal 0.000...<br>chaîne vide ("")<br>chaîne égale à 0 ("0")<br>tableau vide<br>objet vide<br>constante NULL (cf. le type NULL) | FALSE |
| tout le reste | TRUE |

---

Une valeur égale à -1 est convertie en TRUE avec PHP.

---

Inversement, PHP est capable d'opérer les conversions suivantes :

| | **TRUE** | **FALSE** |
|---|---|---|
| **Booléen è Nombre** | 1 | 0 |
| **Booléen è Chaîne** | "1" | "" (chaîne vide) |

Compte tenu de la logique de conversion indiquée précédemment, toute variable peut être testée en tant que booléen (PHP se chargeant de la conversion). Ce fonctionnement est souvent pratique mais peut facilement conduire à des erreurs délicates à déceler.

### c. Types de données spéciaux

#### Le « type » NULL

Ce type est un peu particulier et correspond au type d'une variable utilisée sans jamais avoir été initialisée. Il possède une seule valeur, la valeur NULL définie par la constante NULL (ou null).

En cas de conversion en booléen, la valeur NULL est convertie en FALSE.

#### Le type ressource (

Ce type générique est un peu particulier, et correspond à une référence vers une ressource externe : fichier ouvert, connexion de base de données, etc.

À plusieurs reprises dans cet ouvrage, nous aurons l'occasion de présenter des fonctions qui permettent de manipuler ces données de type ressource.

## 4. Tableaux

### a. Définition

En PHP, un tableau est une collection (liste d'éléments) ordonnée de couples clé/valeur.

La clé peut être de type entier ou de type chaîne. Dans le premier cas, le tableau est dit numérique et la clé est désignée par le terme indice. Dans le deuxième cas le tableau est dit associatif. Les clés ne sont pas forcément consécutives ni ordonnées et un tableau peut comporter des clés entières et des clés de type chaîne.

La valeur associée à la clé peut être de n'importe quel type, et notamment de type tableau ; dans ce cas, le tableau est dit multidimensionnel.

*Exemple*

- Tableau numérique (indices ordonnés consécutifs)

| **Clé/Indice** | **Valeur** |
|---|---|
| 0 | zéro |
| 1 | un |
| 2 | deux |
| 3 | trois |

- Tableau numérique (indices non ordonnés, non consécutifs)

| **Clé/Indice** | **Valeur** |
|---|---|
| 20 | vingt |
| 30 | trente |
| 10 | dix |

- Tableau mixte

| **Clé/Indice** | **Valeur** |
|---|---|
| 0 | zéro |
| zéro | 0 |
| un | 1 |
| 1 | un |
| deux | 2 |
| 2 | deux |
| trois | 3 |
| 3 | trois |

- Tableau multidimensionnel (liste de villes par pays)

| **Clé/Indice** | **Valeur** | |
|---|---|---|
| FRANCE | **Clé/Indice** | **Valeur** |
| | 0 | Paris |
| | 1 | Lyon |
| | 2 | Nantes |
| ITALIE | **Clé/Indice** | **Valeur** |
| | 0 | Rome |
| | 1 | Venise |

### b. Création

Une variable de type tableau peut être définie explicitement grâce à la fonction `array()` ou implicitement en utilisant une notation à crochet ([]).

#### Notation à crochet (

Une variable utilisée pour la première fois avec une notation de la forme `$variable[...]`, est automatiquement créée avec le type tableau.

La même opération effectuée sur une variable déjà définie avec un type scalaire provoque un message d'erreur.

Le contenu d'un tableau peut être ainsi défini par plusieurs affectations du type `$tableau[...] = valeur`.

Avec une affectation du type `$tableau[] = valeur`, PHP recherche le plus grand indice entier utilisé et associe la valeur à l'indice immédiatement supérieur. Si le tableau est vide, l'élément est placé à l'indice 0.

Avec une affectation du type `$tableau[clé] = valeur`, PHP associe la valeur à la clé indiquée (qui peut être de type entier ou de type chaîne).

Les deux notations peuvent être mélangées dans une séquence d'affectation

*Exemple*

```
<?php
$nombres[] = 'zéro'; // => indice 0
$nombres[] = 'un'; // => indice max (0) + 1 = 1
$nombres[] = 'deux'; // => indice max (1) + 1 = 2
$nombres[] = 'trois'; // => indice max (2) + 1 = 3
$nombres[5] = 'cinq'; // => indice 5
$nombres[] = 'six'; // => indice max (5) + 1 = 6
$nombres['un'] = 1; // indice 'un'
$nombres[] = 'sept'; // => indice max (6) + 1 = 7
$nombres[-1] = 'moins un'; // => -1
?>
```

*Résultat*

| Clé/Indice | Valeur |
|---|---|
| 0 | zéro |
| 1 | un |
| 2 | deux |
| 3 | trois |
| 5 | cinq |
| 6 | six |
| un | 1 |
| 7 | sept |
| -1 | moins un |

Ces notations peuvent être utilisées pour construire un tableau multidimensionnel, sous la forme `$tableau[...] = $tableau_intérieur` ou `$tableau[...][...] = valeur`. La première notation permet de stocker un tableau dans un emplacement d'un autre tableau et la deuxième notation, de stocker une valeur directement dans un emplacement situé à l'intérieur d'un autre tableau.

*Exemple*

- Première méthode :

```
<?php
// Création d'un tableau contenant les villes de France
$villes_france[] = 'Paris';
$villes_france[] = 'Lyon';
$villes_france[] = 'Nantes';
// Stockage du tableau des villes de France dans le tableau
// des villes.
$villes['FRANCE'] = $villes_france;
```

```
// Idem avec les villes d'Italie
$villes_italie[] = 'Rome';
$villes_italie[] = 'Venise';
$villes['ITALIE'] = $villes_italie;
?>
```

- Deuxième méthode :

```
<?php
// Stockage direct des villes dans le tableau
//   - pour la France
$villes['FRANCE'][] = 'Paris';
$villes['FRANCE'][] = 'Lyon';
$villes['FRANCE'][] = 'Nantes';
//  - pour l'Italie
$villes['ITALIE'][] = 'Rome';
$villes['ITALIE'][] = 'Venise';
?>
```

*Résultat (dans les deux cas)*

| **Clé/Indice** | **Valeur** | |
|---|---|---|
| FRANCE | **Clé/Indice** | **Valeur** |
| | 0 | Paris |
| | 1 | Lyon |
| | 2 | Nantes |
| ITALIE | **Clé/Indice** | **Valeur** |
| | 0 | Rome |
| | 1 | Venise |

## La fonction array

La fonction array permet de créer un tableau à partir d'une liste d'éléments.

*Syntaxe*

```
tableau array([mixte valeur[, ...]])
ou
tableau array([{chaîne | entier} clé => mixte valeur[, ...]])
```

valeur

Élément du tableau.

clé

Valeur de la clé.

Dans la première syntaxe, les clés/indices ne sont pas spécifiés et c'est un tableau numérique à indices consécutifs commençant à 0 qui est créé : le premier argument de la fonction étant stocké à l'indice 0, le deuxième à l'indice 1, etc.

Dans la deuxième syntaxe, l'indice ou la clé est spécifié(e) par un entier ou par une chaîne et une valeur lui est associée par l'opérateur =>.

Les deux syntaxes peuvent être mélangées. Dans ce cas, lorsque l'indice ou la clé n'est pas spécifié(e), PHP recherche le plus grand indice entier utilisé et associe la valeur à l'indice immédiatement supérieur ; s'il n'existe aucun indice entier, l'élément est placé à l'indice 0.

La fonction array, appelée sans argument, crée un tableau vide.

*Exemple*

```
<?php
$nombres = array('zéro','un','deux','trois',
  5 => 'cinq','six','un' => 1,'sept',-1 => 'moins un');
?>
```

*Résultat*

| Clé/Indice | Valeur |
|---|---|
| 0 | zéro |
| 1 | un |
| 2 | deux |
| 3 | trois |
| 5 | cinq |
| 6 | six |
| un | 1 |
| 7 | sept |
| -1 | moins un |

La fonction array accepte en argument les données de type tableau (soit une variable, soit un appel imbriqué à array) ce qui permet de construire un tableau multidimensionnel.

*Exemple*

```
<?php
// Création d'un tableau contenant les villes de France et
// d'Italie
// Le tableau des villes de France est créé au préalable.
$villes_france = array('Paris','Lyon','Nantes');
// Celui des villes d'Italie est créé par un appel imbriqué
// à array().
$villes = array('FRANCE' => $villes_france,
                'ITALIE' => array('Rome','Venise'));
?>
```

*Résultat*

| Clé/Indice | Valeur | |
|---|---|---|
| FRANCE | **Clé/Indice** | **Valeur** |
| | 0 | Paris |
| | 1 | Lyon |
| | 2 | Nantes |
| ITALIE | **Clé/Indice** | **Valeur** |
| | 0 | Rome |
| | 1 | Venise |

### c. Manipulation

Deux besoins courants existent, relatifs à la manipulation d'un tableau :

- accéder à un élément individuel du tableau ;
- parcourir le tableau.

#### Accéder à un élément individuel du tableau

La notation à crochets est utilisée pour accéder, en lecture ou en écriture, à un élément individuel du tableau :

```
$tableau[{chaîne | entier} clé]
```

`$tableau`

Tableau concerné.

`clé`

Valeur de la clé/indice.

Pour les tableaux multidimensionnels, plusieurs séries de crochets doivent être utilisés.

*Exemple*

```
<?php
$nombres = array('zéro','un','deux','trois',
  5 => 'cinq','six','un' => 1,'sept',-1 => 'moins un');
echo $nombres[1],'<br />';
echo $nombres['un'],'<br />';
$villes = array('FRANCE' => array('Paris','Lyon','Nantes'),
                'ITALIE' => array('Rome','Venise'));
echo $villes['FRANCE'][0],'<br />';
echo $villes['ITALIE'][1],'<br />';
?>
```

*Résultat*

```
un
1
Paris
Venise
```

Le principe de substitution des variables dans les chaînes délimitées par des guillemets fonctionne avec les tableaux. Des accolades sont nécessaires pour délimiter l'expression dans deux cas :

- Pour spécifier une clé de type chaîne exprimée sous la forme d'un littéral : `{$tableau['...']}`.
- Pour un tableau multidimensionnel : `{$tableau[...][...]}`.

*Exemple*

```
<?php
$nombres = array('zéro','un','deux','trois',
  5 => 'cinq','six','un' => 1,'sept',-1 => 'moins un');
echo "\$nombres[1] = $nombres[1]<br />";
echo "\$nombres['un'] = {$nombres['un']}<br />";
$villes = array('FRANCE' => array('Paris','Lyon','Nantes'),
                'ITALIE' => array('Rome','Venise'));
echo "\$villes['FRANCE'][0] = {$villes['FRANCE'][0]}<br />";?>
```

*Résultat*

```
$nombres[1] = un
$nombres['un'] = 1
$villes['FRANCE'][0] = Paris
```

## Parcourir le tableau

De nombreuses méthodes peuvent être utilisées pour parcourir un tableau à l'aide des constructions suivantes :

- la structure de contrôle itérative `for`,
- la structure de contrôle itérative `while`,
- la structure de parcours de tableau `foreach`.

Dans ce livre, nous étudions uniquement l'utilisation de la structure `foreach` qui est sans conteste la méthode la plus simple pour parcourir un tableau. Cette méthode ne nécessite aucune connaissance particulière sur la nature du tableau (numérique, associatif, plage des indices/clés, ...).

*Syntaxe*

```
foreach(tableau as variable_valeur)
```

```
  { instructions }
ou
foreach(tableau as variable_clé => variable_valeur)
  { instructions }
```

La première syntaxe permet de parcourir le tableau du début à la fin ; à chaque itération, la valeur courante du tableau est stockée dans la variable variable_valeur et les instructions entre accolades sont exécutées. Cette syntaxe est suffisante si le traitement n'a pas besoin de faire référence aux valeurs de la clé.

La deuxième syntaxe fonctionne sur le même principe, mais à chaque itération, la clé courante est stockée dans la variable variable_clé et la valeur dans la variable variable_valeur. Cette syntaxe est pratique si le traitement a besoin de faire référence aux valeurs de la clé.

*Exemple*

```
<?php
// Initialisation d'un tableau.
$nombres = array('zéro','un','deux',
                 'zéro' => 0,'un' => 1,'deux' => 2);
// Parcours du tableau avec la première syntaxe.
echo 'Première syntaxe :<br />';
foreach($nombres as $nombre) {
  echo "$nombre<br />";
}
// Parcours du tableau avec la deuxième syntaxe
echo 'Deuxième syntaxe :<br />';
foreach($nombres as $clé => $nombre) {
  echo "$clé => $nombre<br />";
}
?>
```

*Résultat*

```
Première syntaxe :
zéro
un
deux
0
1
2
Deuxième syntaxe :
0 => zéro
1 => un
2 => deux
zéro => 0
un => 1
deux => 2
```

En cas de besoin, l'instruction break (cf. dans ce chapitre - Les bases du langage PHP - Structures de contrôle) peut être utilisée pour interrompre le parcours du tableau avant la fin.

## 5. Opérateurs

### a. L'opérateur d'affectation par valeur

L'opérateur d'affectation est le signe égal (=).

*Syntaxe*

```
$variable = expression;
```

*Exemple*

```
<?php
$nom = 'Olivier';
$indice = 1;
?>
```

Avec cette syntaxe, l'affectation s'effectue par valeur, c'est-à-dire que la valeur de l'expression située à droite du signe égal est copiée dans la variable mentionnée à gauche.

L'opération d'affectation est une expression qui a une valeur égale à la valeur affectée et qui peut être utilisée directement dans

une autre expression. Par exemple, la valeur de l'expression $x=1 est 1 et il est licite d'écrire une instruction du type $y=($x=1)+2 affectant la valeur 3 à $y.

*Exemple*

```
<?php
// Affectation en une instruction de $x et $y.
$y = ($x = 1) + 2;
// Affichage du résultat.
echo "\$x = $x<br />";
echo "\$y = $y<br />";
?>
```

*Résultat*

```
$x = 1
$y = 3
```

Cette technique est très pratique mais peut nuire à la lisibilité du code.

---

Pour tous les opérateurs qui seront étudiés dans ce chapitre, des espaces peuvent ou non être présents autour de l'opérateur.

---

### b. L'opérateur d'affectation par référence

Il est possible de faire une affectation par référence en utilisant l'opérateur &.

*Syntaxe*

```
$variable2 = &$variable1;
```

*Exemple*

```
<?php
$nom = 'Olivier';
$patronyme = &nom;
?>
```

Avec cette syntaxe, la valeur de la variable $variable1 n'est pas copiée dans la variable $variable2. La variable $variable2 fait référence à la variable $variable1 ; les deux variables pointent vers la même zone mémoire et la modification d'une des deux variables se répercute sur l'autre.

*Exemple*

```
<?php
// Initialisation d'une variable.
$nom = 'Olivier';
// Affectation dans une autre variable par référence.
$patronyme = &$nom;
// Affichage du résultat.
echo "<b>Initialement :</b><br />";
echo "\$nom = $nom<br />";
echo "\$patronyme = $patronyme<br />";
// Modification de la première variable.
$nom = 'Heurtel';
// Affichage du résultat.
echo "<b>Après modification de \$nom :</b><br />";
echo "\$nom = $nom<br />";
echo "\$patronyme = $patronyme<br />";
?>
```

*Résultat*

```
Initialement :
$nom = Olivier
$patronyme = Olivier
Après modification de $nom :
$nom = Heurtel
$patronyme = Heurtel
```

### c. Les opérateurs arithmétiques

Les opérateurs arithmétiques sont les suivants :

| **Opération** | **Opérateur** | **Exemple($x=13 et $y=8)** |
|---|---|---|
| Somme | `+` | `echo $x + $y;=> 21` |
| Soustraction | `-` | `echo $x - $y;=> 5` |
| Multiplication | `*` | `echo $x * $y;=> 104` |
| Division | `/` | `echo $x / $y;=> 1.625` |
| Modulo (reste de la division entière du premier opérande par le deuxième) | `%` | `echo $x % $y;=> 5` |
| Opposé | `-` | `echo -$x;=> -13` |
| Préincrémentation (incrémente la variable <u>avant</u> de retourner la valeur de la variable) | `++` avant l'opérande | `echo ++$x;=> 14` |
| Postincrémentation (incrémente la variable <u>après</u> avoir retourné la valeur de la variable) | `++` après l'opérande | `echo $x++;=> 13`<br>`echo $x;=> 14` |
| Prédécrémentation (décrémente la variable <u>avant</u> de retourner la valeur de la variable) | `--` avant l'opérande | `echo --$x;=> 12` |
| Postdécrémentation (décrémente la variable <u>après</u> avoir retourné la valeur de la variable) | `--` après l'opérande | `echo $x--;=> 13`<br>`echo $x;=> 12` |

### d. L'opérateur de chaîne

Le seul opérateur de chaîne est l'opérateur de concaténation égal au point (.).

*<u>Syntaxe</u>*

```
chaîne1.chaîne2;
```

Cet opérateur retourne une chaîne égale à la première chaîne immédiatement suivie de la deuxième ; aucun séparateur n'est mis entre les deux chaînes.

*<u>Exemple</u>*

```
<?php
// Utilisation de l'opérateur de concaténation avec des
// variables et des expressions littérales.
$prénom = 'Olivier';
$nom = 'Heurtel';
echo $nom.', '.$prénom.'<br />';
?>
```

*<u>Résultat</u>*

```
Heurtel, Olivier
```

### e. Les opérateurs de comparaison

Les opérateurs de comparaison sont les suivants :

| **Opération** | **Opérateur** | **Exemple($x=13, $y=8, $z="8")** |
|---|---|---|
| Égalité | `==` | `$x == $y => FALSE`<br>`$y == $z => TRUE` |
| Égalité **et** types identiques | `===` | `$x === $y => FALSE` |

| | | |
|---|---|---|
| | | $y === $z => FALSE |
| Différent | != | $x != $y => TRUE<br>$y != $z => FALSE |
| Différent **ou** types différents | !== | $x !== $y => TRUE<br>$y !== $z => TRUE |
| Inférieur | < | $x < $y => FALSE<br>$y < $x => TRUE<br>$y < $z => FALSE |
| Inférieur ou égal | <= | $x <= $y => FALSE<br>$y <= $x => TRUE<br>$y <= $z => TRUE |
| Supérieur | > | $x > $y => TRUE<br>$y > $x => FALSE<br>$y > $z => FALSE |
| Supérieur ou égal | >= | $x >= $y => TRUE<br>$y >= $x => FALSE<br>$y >= $z => TRUE |

Ne confondez pas l'opérateur d'affectation (=) avec l'opérateur de comparaison (==).

### f. Les opérateurs logiques

Les opérateurs logiques sont les suivants :

| **Opération** | **Opérateur** | **Exemple** |
|---|---|---|
| Et logique | and<br>&& | TRUE and TRUE => TRUE<br>TRUE and FALSE => FALSE<br>FALSE and FALSE => FALSE |
| Ou logique | or<br>\|\| | TRUE or TRUE => TRUE<br>TRUE or FALSE => TRUE<br>FALSE or FALSE => FALSE |
| Ou logique exclusif (FALSE si les deux opérandes sont TRUE) | xor | TRUE xor TRUE => FALSE<br>TRUE xor FALSE => FALSE<br>FALSE xor FALSE => FALSE |
| Non logique | ! | ! TRUE => FALSE<br>! FALSE => TRUE |

Les opérateurs and et && ainsi que or et || sont identiques mais n'ont pas la même précédence.

### g. L'opérateur ternaire

Un autre opérateur conditionnel, l'opérateur ternaire ?, fonctionne comme dans le langage C.

*Syntaxe*

```
expression1?expression2:expression3
```

Cette instruction retourne la valeur de `expression2` si `expression1` est évaluée à `TRUE` et la valeur de `expression3` si `expression1` est évaluée à `FALSE`. Si `expression1` n'est pas de type booléen, une conversion est effectuée selon les règles décrites précédemment (cf. dans ce chapitre - Les bases du langage PHP - Types de données).

*Exemple*

```
<?php
// Initialisation d'une variable.
$nom = '';
// Affichage d'un message dépendant de la valeur de $nom.
echo 'Bonjour '.(($nom=='')?'inconnu':$nom).' ! <br />';
// Affectation d'une valeur à la variable $nom.
$nom = 'Olivier';
// nouvelle tentative
echo 'Bonjour '.(($nom=='')?'inconnu':$nom).' ! <br />';
?>
```

*Résultat*

```
Bonjour inconnu !
Bonjour Olivier !
```

### h. Les opérateurs combinés

Les opérateurs somme (+), différence (-), multiplication (*), division (/), module (%) et concaténation (.) peuvent être combinés avec l'opérateur d'affectation (=) selon la syntaxe suivante :

| **Syntaxe** | **Équivalent à** |
|---|---|
| `$variable += expression` | `$variable = $variable + expression` |
| `$variable -= expression` | `$variable = $variable - expression` |
| `$variable *= expression` | `$variable = $variable * expression` |
| `$variable /= expression` | `$variable = $variable / expression` |
| `$variable %= expression` | `$variable = $variable % expression` |
| `$variable .= expression` | `$variable = $variable . expression` |

### i. Précédence des opérateurs

La précédence des opérateurs désigne l'ordre dans lesquels les opérateurs sont traités dans une expression complète.

Comme dans tous les langages, les parenthèses peuvent être utilisées pour modifier l'ordre dans lequel les opérations sont traitées. Dans la pratique, n'hésitez pas à utiliser les parenthèses pour éviter tout problème et améliorer la lisibilité des expressions.

La précédence des opérateurs est la suivante, du moins prioritaire (traité en dernier) au plus prioritaire (traité en premier) :

**Opérateur**

`or`

`xor`

`and`

`= += -= *= /= %= .=`

`?:`

`||`

`&&`

`== != ===`

`< <= > >=`

```
+ - .

* / %

! ++ -- (int) (double) (string) (array) (object)
```

## 6. Structures de contrôle

### a. La structure if

La structure de contrôle `if` permet une exécution conditionnelle d'instructions.

*Syntaxe*

```
if (condition) {
  instructions;
[ } elseif (condition) {
  instructions; ]
[ ... ]
[ } else {
  instructions; ]
}
```

Les conditions des clauses `if` et `elseif` sont testées séquentiellement. Si la condition est vraie alors les instructions associées de la clause sont exécutées. Si aucune des conditions n'est vraie, les instructions éventuelles définies dans la clause `else` sont exécutées.

Si les expressions définissant les conditions ne sont pas de type booléen, une conversion est effectuée selon les règles décrites précédemment (cf. dans ce chapitre - Les bases du langage PHP - Types de données).

*Exemple*

```
<?php
// Structure if / elseif / else
$nom = 'Olivier';
$âge = NULL;
if ($nom == NULL) {
  echo "Bonjour inconnu !<br />";
} elseif ($âge == NULL) {
  echo "Bonjour $nom ! Je ne connais pas votre âge.<br />";
} else {
  echo "Bonjour $nom ! Vous avez $âge ans.<br />";
}
?>
```

*Résultat*

```
Bonjour Olivier ! Je ne connais pas votre âge.
```

Une deuxième syntaxe peut être utilisée pour écrire une structure de contrôle sur plusieurs blocs PHP entre lesquels du code HTML est intercalé :

```
<?php if (condition): ?>
  code_HTML
[ <?php elseif (condition): ?>
  code_HTML ]
[ ... ]
[ <?php else: ?>
  code_HTML ]
<?php endif; ?>
```

Le principe d'analyse de la structure `if - elseif - else` est le même qu'avec la première syntaxe, mais au lieu d'exécuter des instructions PHP, le moteur incorpore dans le résultat le code HTML associé à la condition.

*Exemple*

```
<?php
// Un peu d'aléatoire pour définir les variables $nom et $âge.
$nom = rand(0,1)?'Olivier':NULL;
$âge = rand(0,1)?rand(7,77):NULL;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
    <style type="text/css" media="all">
      .ko {font-weight: bold; color: red;}
      .ok {font-weight: bold; color: green;}
    </style>
  </head>
  <body>
    <div>
    <?php if ($nom == NULL) : // condition PHP ?>
       <!-- Code HTML -->
       Bonjour inconnu !<br />
    <?php elseif ($âge == NULL) : // suite de la condition ?>
       <!-- Code HTML -->
       Je connais votre <span class="ok">nom</span>
       mais pas votre <span class="ko">âge</span>.<br />
    <?php else : // suite de la condition PHP ?>
       <!-- Code HTML -->
       Je connais votre <span class="ok">nom</span>
       et votre <span class="ok">âge</span>,
       mais je ne dirai rien !<br />
    <?php endif; // fin de la condition PHP ?>
    </div>
  </body>
</html>
```

*Résultat (dépend de l'affectation aléatoire des variables)*

```
Je connais votre nom mais pas votre âge.
```

Cette syntaxe est réellement très pratique pour faire une construction conditionnelle d'une page HTML, en évitant la lourdeur de l'utilisation d'un seul bloc PHP générant tout le code HTML avec l'instruction `echo`.

### b. La structure switch

La structure de contrôle `switch`, équivalente à une construction `if - elseif` multiple, est utilisée pour la comparaison du résultat d'une expression avec plusieurs résultats.

Comme l'instruction `if`, cette instruction possède deux syntaxes :

*Première syntaxe*

```
switch (expression_test) {
  case expression:
    instructions;
    [break;]
  [ case expression:
    instructions;
    [break;] ]
  [ ... ]
  [ default:
    instructions;
    [break;] ]
}
```

`expression_test` est une expression qui est évaluée une fois et dont la valeur est comparée séquentiellement avec les expressions des clauses `case`. Si `expression_test` est égale à l'expression de la clause `case` alors les instructions associées sont exécutées et les comparaisons se poursuivent s'il n'y a pas d'instruction `break`. Si aucune égalité n'est trouvée, les instructions éventuellement définies dans la clause `default` sont exécutées.

 Pour interrompre l'exécution de l'instruction `switch` et l'évaluation des clauses `case`, il faut utiliser l'instruction `break`.

```
<?php
$nom = rand(0,1)?'Olivier':NULL;
switch ($nom) {
   case NULL :
      echo 'Bonjour inconnu ! ',
           'Je vais vous appeler Olivier.<br />';
      $nom = 'Olivier';
      break;
   case 'Olivier' :
      echo "Bonjour Maître $nom !<br />";
      break;
```

```
   default :
      echo "Bonjour élève $nom !<br />";
}
?>
```

*Résultat (dépend de l'affectation aléatoire des variables)*

```
Bonjour inconnu ! Je vais vous appeler Olivier.
```

Comme pour l'instruction `if`, il existe une deuxième syntaxe qui permet d'imbriquer du code HTML dans une structure de contrôle `switch` :

```
<?php switch (expression_test):
  case (expression): ?>
    code_HTML
  [ <?php case (expression): ?>
    code_HTML ]
  [ ... ]
  [ <?php default: ?>
    code_HTML ]
<?php endswitch; ?>
```

Le premier `case` doit être écrit dans le bloc PHP `switch`.

*Exemple*

```
<?php
$langue = 'fr';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
    <style type="text/css" media="all">
    .en {font-weight: bold; color: green;}
    .sp {font-weight: bold; color: orange;}
    .fr {font-weight: bold; color: blue;}
    .inconnu {font-weight: bold; color: red;}
    </style>
  </head>
  <body>
    <div>
    <?php switch ($langue) : // switch
            case 'en' :      // premier case
    ?>
       <!-- Code HTML -->
       Hello <span class="en">my friend</span> !<br />
    <?php   break;           // break premier case ?>
    <?php   case 'sp' :      // deuxième case ?>
       <!-- Code HTML -->
       ¡ Buenos dias <span class="sp">amigo</span> !<br />
    <?php   break;           // break deuxième case ?>
    <?php   case 'fr' :      // troisième case ?>
       <!-- Code HTML -->
       Salut <span class="fr">mon pote</span> !<br />
    <?php   break;           // break troisième case ?>
    <?php   default :        // défaut ?>
       <!-- Code HTML -->
       <span class="inconnu">?????</span>
    <?php endswitch;         // fin du switch ?>
    </div>
  </body>
</html>
```

*Résultat*

Salut **mon pote** !

### c. La structure while

La structure de contrôle `while` permet d'exécuter en boucle une série d'instructions tant qu'une condition est vraie.

*Syntaxe*

```
while (condition) {
  instructions;
}
```

Les instructions à l'intérieur de la boucle sont exécutées tant que la condition de la clause `while` est vraie. Si l'expression définissant la condition n'est pas de type booléen, une conversion est effectuée selon les règles décrites précédemment (cf. dans ce chapitre - Les bases du langage PHP - Types de données).

*Exemple*

```
<?php
// Initialiser deux variables.
$nom = 'OLIVIER';
$longueur = 7;
// Initialiser un indice.
$indice = 0;
// Tant que l'indice est inférieur à la longueur de la chaîne
while ($indice < $longueur) {
  // Afficher le caractère correspondant à l'indice suivi
  // d'un point.
  echo "$nom[$indice].";
  // Incrémenter l'indice
  $indice++;
}
?>
```

*Résultat*

```
O.L.I.V.I.E.R.
```

Si la condition est fausse à la première itération, les instructions situées à l'intérieur de la boucle ne sont jamais exécutées. Si la condition n'est jamais fausse, les instructions à l'intérieur de la boucle sont exécutées sans fin (pas tout à fait puisque le temps d'exécution d'un script est limité par la directive de configuration `max_execution_time`).

Comme pour les structures conditionnelles, une deuxième syntaxe permet d'imbriquer du code HTML dans une structure de contrôle `while` :

```
<?php while (condition): ?>
  code_HTML
<?php endwhile; ?>
```

*Exemple*

```
<?php
// Initialiser deux variables.
$numéro = 0;
$nombre = 5;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
  </head>
  <body>
    <form action="">
       <div>
       Indiquez vos cinq compétences principales :<br />
       <?php while($numéro++ < $nombre) : // boucle PHP ?>
          <!-- Code HTML -->
          <input type="text" /><br />
       <?php endwhile; // fin de la boucle PHP ?>
       <input type="submit" value = "OK" /><br />
       </div>
    </form>
  </body>
</html>
```

*Résultat*

Indiquez vos cinq compétences principales :

OK

### d. La structure do ... while

La structure de contrôle `do ... while` permet d'exécuter en boucle une série d'instructions tant qu'une condition est vraie.

*Syntaxe*

```
do {
  instructions;
} while (expression);
```

Les instructions à l'intérieur de la boucle sont exécutées tant que la condition de la clause `while` est vraie. À la différence de la structure `while`, la condition est testée à la fin de la boucle ; les instructions `instructions` sont donc forcément exécutées au moins une fois. Si l'expression définissant la condition n'est pas de type booléen, une conversion est effectuée selon les règles décrites précédemment (cf. dans ce chapitre - Les bases du langage PHP - Types de données).

*Exemple*

```
<?php
// Initialiser deux variables.
$nom = 'OLIVIER';
$longueur = 7;
// Initialiser un indice.
$indice = 0;
// Tant que l'indice est inférieur à la longueur de la chaîne
do {
  // Afficher le caractère correspondant à l'indice suivi
  // d'un point.
  echo "$nom[$indice].";
  // Incrémenter l'indice
  $indice++;
} while ($indice < $longueur);
?>
```

*Résultat*

```
O.L.I.V.I.E.R.
```

Contrairement aux autres structures de contrôle, une seule syntaxe est disponible.

### e. La structure for

La structure de contrôle `for`, comme dans le langage C, permet d'exécuter des instructions de manière itérative en contrôlant les itérations à l'aide de trois expressions.

*Syntaxe*

```
for (expression1; expression2; expression3) {
  instructions;
}
```

Le principe de fonctionnement de cette structure de contrôle est le suivant :

- `expression1` est exécutée une fois au démarrage de la boucle.
- `expression2` est exécutée, et le résultat est évalué comme booléen, avant chaque itération (dont la première). Si le résultat est évalué à `TRUE` les instructions `instructions` sont exécutées ; si le résultat est évalué à `FALSE`, la boucle s'arrête et le contrôle est passé à la première instruction qui suit la construction.
- `expression3` est exécutée à la fin de chaque itération.

Dans la grande majorité des cas, l'instruction `for` est employée de la manière suivante :

- `expression1` initialise un compteur.
- `expression2` teste la valeur du compteur.
- `expression3` incrémente la valeur du compteur.

Cette utilisation permet d'exécuter des instructions un nombre donné de fois.

*Exemple*

```
<?php
// Utilisation de la structure for pour parcourir un tableau
// à indices entiers consécutifs
// Initialisation du tableau.
$couleurs = array('bleu','blanc','rouge');
$nombre = 3;
// Boucle utilisant un indice $i qui démarre à 0 ($i = 0)
// qui est incrémenté d'une unité à chaque itération ($i++) ;
// la boucle se poursuit tant que l'indice est inférieur au
// nombre d'éléments présents dans le tableau ($i < $nombre).
for ($i = 0; $i < $nombre; $i++) {
  echo "$couleurs[$i]<br />";
};
?>
```

*Résultat*

```
bleu
blanc
rouge
```

Une deuxième syntaxe permet d'imbriquer du code HTML dans une structure de contrôle `for` :

```
<?php for (expression1; expression2; expression3) : ?>
    code_HTML
<?php endfor; ?>
```

L'exemple fourni pour la structure `while` peut être écrit avec une structure `for`.

*Exemple*

```
<?php
// Initialiser deux variables.
$numéro = 0;
$nombre = 5;
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Exemple de page PHP</title>
  </head>
  <body>
    <form action="">
       <div>
       Indiquez vos cinq compétences principales :<br />
       <?php // boucle PHP
       for($numéro = 1; $numéro <= $nombre; $numéro++):
       ?>
          <!-- Code HTML -->
          <input type="text" /><br />
       <?php endfor; // fin de la boucle PHP ?>
       <input type="submit" value = "OK" /><br />
       </div>
    </form>
  </body>
</html>
```

### f. Les instructions continue et break

L'instruction `continue` peut être utilisée dans toutes les structures de contrôle itératives pour interrompre l'itération en cours et passer à l'itération suivante.

L'instruction `break` peut être utilisée dans toutes les structures de contrôle itératives pour interrompre la boucle en cours. L'instruction `break` peut aussi être utilisée dans une structure de contrôle `switch`.

*Syntaxe*

```
continue[ n];
continue[(n)];
break[ n];
break[(n)];
```

Les deux instructions acceptent un paramètre qui indique de combien de niveau(x) remonter si des structures de contrôle itératives sont imbriquées. Par défaut, les instructions remontent d'un niveau.

*Exemple*

```
<?php
$couleurs = array('bleu','invisible','blanc','rouge');
for ($i = 0; $i <= 3; $i++) {
  // Passer à l'itération suivante pour
  // la couleur "invisible"
  if ($couleurs[$i] == 'invisible') {
    continue;
  }
  echo "$couleurs[$i] ";
}
echo '<br />';
for ($i = 0; $i <= 3; $i++) {
  // Interrompre la boucle à la couleur "invisible"
  if ($couleurs[$i] == 'invisible') {
    break;
  }
  echo "$couleurs[$i] ";
}
;?>
```

*Résultat*

```
bleu blanc rouge
bleu
```

## 7. Inclure un fichier

### a. Fonctionnement

Les fonctions `include`, `include_once`, `require` et `require_once` permettent d'inclure un fichier dans un script PHP.

*Syntaxe*

```
entier include(chaîne fichier)
entier include_once(chaîne fichier)require(chaîne fichier)require_once(chaîne fichier)
```

`fichier`

Nom du fichier à inclure (peut être indiqué avec un chemin absolu ou relatif).

 Dans le fichier `php.ini`, la directive `include_path` permet de définir des chemins de recherche pour l'inclusion des fichiers.

Les fonctions `include` et `include_once` retournent 1 en cas de succès et `FALSE` en cas d'erreur. Les fonctions `require` et `require_once` n'ont pas de code de retour.

En cas d'erreur, les fonctions `include` et `include_once` génèrent une simple erreur de niveau `E_WARNING` qui n'interrompt pas l'exécution du script. Ce n'est pas le cas des fonctions `require` et `require_once` qui provoquent alors une erreur fatale interrompant l'exécution du script.

Le fichier inclus peut contenir du code HTML, du code PHP ou les deux. Le code PHP inclut doit être écrit entre les balises habituelles. Le code HTML présent dans le fichier inclus est intégré tel quel dans la page envoyée au navigateur, comme s'il était présent dans le script appelant. Le code PHP présent dans le fichier inclus est exécuté comme s'il était présent dans le script appelant.

Après l'inclusion tout se passe comme s'il n'y avait qu'un seul script. En conséquence, les variables et constantes définies dans le fichier inclus sont utilisables dans le script appelant et réciproquement.

Il est possible d'inclure plusieurs fichiers dans un script, ou d'imbriquer les inclusions (inclure un fichier qui lui même inclut un autre fichier).

Avec les fonctions `include` et `require`, le processus d'inclusion est répété plusieurs fois si le même fichier est inclus plusieurs fois.

Dans certains cas, cette situation peut être indésirable et involontaire, notamment lorsqu'un fichier est inclus une première fois directement dans un script et une deuxième fois indirectement par l'inclusion d'un autre fichier.

Ce comportement peut être évité en utilisant les fonctions `include_once` et `require_once` qui garantissent qu'un fichier ne sera inclus qu'une fois même s'il est appelé plusieurs fois.

 L'extension du fichier à inclure est parfaitement libre. Il n'est notamment pas obligatoire d'utiliser l'extension `.php` pour inclure du code PHP : le fichier inclus n'est pas directement exécuté par le moteur PHP (c'est le script appelant qui l'est). Pour les fichiers inclus qui ne peuvent pas ou ne doivent pas être exécutés directement, il est alors possible d'utiliser une autre extension (`.inc` par exemple).

*Exemple : script principal*

```
<?php
// Inclusion d'un fichier
include('commun.inc');
// Déclaration d'une variable $x dans le script principal.
$x = 1;
// Affichage de la variable $x.
echo "Valeur de \$x dans le script principal : $x<br />";
// Affichage de la variable $y (définie dans le fichier
// inclus).
echo "Valeur de \$y dans le script principal : $y<br />";
?>
```

*Fichier inclus commun.inc*

```
<!-- Démarrage du fichier d'inclusion en mode HTML (ouverture
---- d'une balise <b> qui est fermée à la fin du fichier -->
<b>Début du fichier commun.inc<br />
<?php // un peu de code PHP
// Déclaration d'une variable $y dans le script inclus.
$y = 2;
// Affichage de la variable $y.
echo "Valeur de \$y dans le fichier inclus : $y<br />";
?>
Fin du fichier commun.inc</b><br />
```

*Résultat*

```
Début du fichier commun.inc
Valeur de $y dans le fichier inclus : 2
Fin du fichier commun.inc
Valeur de $x dans le script principal : 1
Valeur de $y dans le script principal : 2
```

### b. Utilisation

La technique d'inclusion est pratique pour deux grands types d'utilisation :

- Inclure des définitions statiques : constantes, définitions de fonctions ou de classes. Dans ce cas, il faut plutôt utiliser les fonctions `include_once` ou `require_once` afin d'éviter une éventuelle double inclusion (qui provoquerait une erreur en ce qui concerne la définition des fonctions).

- Inclure du code PHP ou HTML dynamique qui s'exécute effectivement au moment de l'inclusion : section HTML commune à plusieurs pages (en-tête, pied de page) ou code commun à plusieurs pages (bien que dans cette hypothèse, la définition d'une fonction soit plus pertinente). Dans ce cas, il faut plutôt utiliser les fonctions `include` ou `require` afin de garantir que l'inclusion se produit bien à chaque appel.

 La création de fonctions personnalisées est étudiée dans le chapitre Écrire des fonctions et des classes PHP.

*Exemple*

- Fichier de définition de constantes (`constantes.inc`)

```
<?php
// Définition des constantes
// par exemple, le nom du site.
DEFINE(NOM_SITE,'monSite.com');
```

```
?>
```

- Fichier contenant le début de chaque page (`debut.inc`)

```
<?php
// Inclusion du fichier des constantes.
include_once('constantes.inc');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title><?php echo NOM_SITE; ?></title></head>
<body>
```

- Fichier contenant la fin de chaque page (`fin.inc`)

```
</body>
</html>
```

- Script d'une page

```
<?php
// Inclusion du début de la page.
include('debut.inc');
?>
<p>Contenu de la page ...</p>
<?php
// Inclusion de la fin de la page
include('fin.inc');
?>
```

*Résultat (source de la page dans le navigateur)*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>monSite.com</title></head>
<body>
<p>Contenu de la page ...</p>
</body>
</html>
```

## 8. Interrompre le script

Les instructions `exit` et `die` permettent d'interrompre l'exécution du script (`die` est un alias d'`exit`).

```
exit[(chaîne message)];
exit[(entier statut)];
die[(chaîne message)];
die[(entier statut)];
```

`message`

Message à afficher avant d'interrompre le script.

`statut`

Statut de retour (entier entre 1 et 254).

Le script s'interrompt brutalement et l'affichage est laissé "en l'état". La page HTML envoyée au navigateur peut donc être incohérente ou vide.

Si l'instruction est appelée dans un fichier inclus, c'est le script principal qui est interrompu.

*Exemple (sans message)*

```
<?php
// Générer le début de la page.
echo 'Bonjour ';
// Une condition n'est pas vérifiée, interrompre le script.
if ($nom == NULL) {
  exit(1);  // pas de message ...
}
```

```
// Poursuivre la génération de la page.
echo $nom;
?>
```

*Résultat*

```
Bonjour
```

*Exemple (avec message)*

```
<?php
// Générer le début de la page.
echo 'Bonjour ';
// Une condition n'est pas vérifiée, interrompre le script.
if ($nom == NULL) {
  exit('<b>Utilisateur inconnu. Impossible de continuer.</b>');
}
// Poursuivre la génération de la page.
echo $nom;
?>
```

*Résultat*

Bonjour **Utilisateur inconnu. Impossible de continuer.**

# Préambule

L'objectif de ce chapitre est de présenter les fonctions les plus utiles dans le cadre du développement d'un site Web.

PHP propose de nombreuses fonctions ; la description de chaque fonction est accessible en ligne sur le site **www.php.net**.

# Manipuler les constantes, les variables et les types de données

## 1. Constantes

PHP propose un certain nombre de fonctions utiles sur les constantes :

| Nom | Rôle |
|---|---|
| `defined` | Indique si une constante est définie ou non. |
| `constant` | Retourne la valeur d'une constante. |

### defined

La fonction `defined` permet de savoir si une constante est définie ou non.

*Syntaxe*

```
booléen defined(chaîne nom)
```

`nom`

Nom de la constante.

La fonction `defined` retourne `TRUE` si la constante est définie et `FALSE` dans le cas contraire.

*Exemple*

```
<?php
// Tester si la constante CONSTANTE est définie.
$ok = defined('CONSTANTE');
if ($ok) {
  echo 'CONSTANTE est définie.<br />';
} else {
  echo 'CONSTANTE n\'est pas définie.<br />';
};
// Définir la constante CONSTANTE
define('CONSTANTE','valeur de CONSTANTE');
// Tester si la constante CONSTANTE est définie.
$ok = defined('CONSTANTE');
if ($ok) {
  echo 'CONSTANTE est définie.<br />';
} else {
  echo 'CONSTANTE n\'est pas définie.<br />';
};
?>
```

*Résultat*

```
CONSTANTE n'est pas définie.
CONSTANTE est définie.
```

### constant

La fonction `constant` retourne la valeur d'une constante dont le nom est passé en paramètre.

*Syntaxe*

```
mixte constant(chaîne nom)
```

`nom`

Nom de la constante.

Cette fonction est pratique pour récupérer la valeur d'une constante dont le nom n'est pas connu a priori.

*Exemple*

```
<?php
// Définir le nom de la constante dans une variable.
```

```
$nomConstante = 'CONSTANTE';
// Définir la valeur de la constante.
define($nomConstante,'valeur de CONSTANTE');
// Afficher la valeur de la constante.
echo $nomConstante,' = ',constant($nomConstante);
?>
```

*Résultat*

```
CONSTANTE = valeur de CONSTANTE
```

D'autres fonctions permettent de connaître le type d'une constante (cf. dans ce chapitre - Manipuler les constantes, les variables et les types de données - Types de données).

## 2. Variables

PHP propose un certains nombre de fonctions utiles sur les variables :

| Nom | Rôle |
|---|---|
| isset | Indique si une variable est définie ou non. |
| empty | Indique si une variable est vide ou non. |
| unset | Supprime une variable. |
| var_dump | Affiche des informations sur une variable (type et valeur). |

**isset**

La fonction isset permet de tester si une variable est définie ou non.

*Syntaxe*

```
booléen isset(mixte variable)
```

variable

Variable à tester.

La fonction isset retourne TRUE si la variable est définie et FALSE dans le cas contraire.

Une variable est considérée comme non définie si elle n'a pas été affectée ou si elle contient NULL.

*Exemple*

```
<?php
// Test d'une variable non initialisée.
$est_définie = isset($variable);
echo '$variable non initialisée<br />';
if ($est_définie) {
  echo '=> $variable est définie.<br />';
} else {
  echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant une chaîne vide.
$variable = '';
$est_définie = isset($variable);
echo '$variable = \'\'<br />';
if ($est_définie) {
  echo '=> $variable est définie.<br />';
} else {
  echo '=> $variable n\'est pas définie.<br />';
}
// Test d'une variable contenant une chaîne non vide.
$variable = 'x';
$est_définie = isset($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($est_définie) {
  echo '=> $variable est définie.<br />';
} else {
```

```
  echo '=> $variable n\'est pas définie.<br />';
}
?>
```

*Résultat*

```
$variable non initialisée
=> $variable n'est pas définie.
$variable = ''
=> $variable est définie.
$variable = 'x'
=> $variable est définie.
```

### empty

La fonction empty permet de tester si une variable est vide ou non.

*Syntaxe*

```
booléen empty(mixte variable)
```

variable

Variable à tester.

La fonction empty retourne TRUE si la variable est vide et FALSE dans le cas contraire.

Une variable est considérée comme vide si elle n'a pas été affectée ou si elle contient une chaîne vide (""), une chaîne égale à 0 ("0"), 0, NULL ou FALSE.

*Exemple*

```
<?php
// Test d'une variable non initialisée.
$est_vide = empty($variable);
echo '$variable non initialisée<br />';
if ($est_vide) {
  echo '=> $variable est vide.<br />';
} else {
  echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant une chaîne vide.
$variable = '';
$est_vide = empty($variable);
echo '$variable = \'\'<br />';
if ($est_vide) {
  echo '=> $variable est vide.<br />';
} else {
  echo '=> $variable n\'est pas vide.<br />';
}
// Test d'une variable contenant une chaîne non vide.
$variable = 'x';
$est_vide = empty($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($est_vide) {
  echo '=> $variable est vide.<br />';
} else {
  echo '=> $variable n\'est pas vide.<br />';
}
?>
```

*Résultat*

```
$variable non initialisée
=> $variable est vide.
$variable = ''
=> $variable est vide.
$variable = 'x'
=> $variable n'est pas vide.
```

### unset

La fonction unset permet de supprimer une variable.

*Syntaxe*

```
unset(mixte variable[, ...])
```

`variable`

Variable à supprimer (éventuellement plusieurs, séparées par une virgule).

La fonction `unset` accepte une liste de variables.

Après suppression, la variable se trouve dans le même état que si elle n'avait jamais été affectée. L'utilisation de la fonction `isset` sur une variable supprimée retourne `FALSE`.

*Exemple*

```
<?php
// Définir une variable.
$variable = 1;
// Afficher la variable et tester si elle est définie.
$est_définie = isset($variable);
echo '$variable = ',$variable,'<br />';
if ($est_définie) {
  echo '=> $variable est définie.<br />';
} else {
  echo '=> $variable n\'est pas définie.<br />';
}
// Supprimer la variable.
unset($variable);
// Afficher la variable et tester si elle est définie.
$est_définie = isset($variable);
echo '$variable = ',$variable,'<br />';
if ($est_définie) {
  echo '=> $variable est définie.<br />';
} else {
  echo '=> $variable n\'est pas définie.<br />';
}
?>
```

*Résultat*

```
$variable = 1
=> $variable est définie.
$variable =
=> $variable n'est pas définie.
```

### var_dump

La fonction `var_dump` affiche des informations sur une variable (type et contenu).

*Syntaxe*

```
var_dump(mixte variable)
```

`variable`

Variable à afficher.

La fonction `var_dump` est surtout intéressante lors des phases de mise au point.

*Exemple*

```
<?php
// Variable non initialisée.
var_dump($variable);
// Variable initialisée avec un nombre entier.
$variable = 10;
echo '<br />';
var_dump($variable);
// Variable initialisée avec un nombre décimal.
$variable = 3.14;
echo '<br />';
var_dump($variable);
// Variable initialisée avec une chaîne.
$variable = 'abc';
echo '<br />';
```

```
var_dump($variable);
?>
```

*Résultat*

```
NULL
int(10)
float(3.14)
string(3) "abc"
```

Pour une variable non initialisée, `var_dump` retourne `NULL`. Pour un nombre, `var_dump` indique le type (`int` = entier, `float` = nombre décimal) suivi de la valeur entre parenthèses. Pour une chaîne, `var_dump` indique le type (`string`) suivi de la longueur entre parenthèses puis de la valeur entre guillemets.

PHP propose aussi les fonctions `print_r` et `var_export` similaires à la fonction `var_dump`. La fonction `print_r` affiche ou retourne le contenu de la variable sous une forme plus lisible, sans mention du type de données. La fonction `var_export` affiche ou retourne une chaîne donnant un code PHP de définition de la variable ; cette fonction est nouvelle en version 5.

---

Dans la section Types de données, nous étudierons d'autres fonctions qui permettent de déterminer le type d'une variable et d'effectuer des conversions de type (nombre en chaîne, chaîne en nombre, etc.).

---

## 3. Types de données

### a. Conversions

PHP est capable d'effectuer des conversions automatiques implicites de type, selon les règles présentées dans le chapitre Introduction à PHP - Les base du langage PHP.

Lorsqu'une valeur/expression est affectée à une variable, la variable devient du type de la valeur/expression.

Pour déterminer le type d'une expression composée d'opérandes de types différents, PHP évalue (mais ne convertit pas) les opérandes en fonction des opérateurs traités dans l'ordre de précédence (cf. chapitre Introduction à PHP - Les base du langage PHP). Par exemple, les deux opérandes utilisés dans une addition sont évalués en nombre alors que deux opérandes utilisés avec l'opérateur de concaténation sont évalués en chaîne.

*Exemple*

```
<?php
$nombre = 123;
$chaîne = "456abc";
echo '$nombre + $chaîne = ';
var_dump($nombre + $chaîne);
echo '<br />';
echo '$nombre . $chaîne = ';
var_dump($nombre . $chaîne);
echo '<br />';
echo '$nombre = ';
var_dump($nombre);
echo '<br />';
echo '$chaîne = ';
var_dump($chaîne);
?>
```

*Résultat*

```
$nombre + $chaîne = int(579)
$nombre . $chaîne = string(9) "123456abc"
$nombre = int(123)
$chaîne = string(6) "456abc"
```

Sur le premier exemple, la variable `$chaîne` a été évaluée en nombre pour être du type attendu par l'opérateur « + » alors que dans le deuxième exemple, c'est `$nombre` qui a été évaluée en chaîne pour être du type attendu par l'opérateur « . » (concaténation). Par contre, les deux derniers affichages montrent que les variables en question n'ont pas été converties lors des opérations : elles conservent leur type respectif initial.

En complément, PHP propose une notation et une fonction pour effectuer une conversion manuelle explicite.

**Notation**

La notation consiste à indiquer le nom du type souhaité entre parenthèses devant l'expression à convertir. Les valeurs autorisées sont les suivantes :

| Notation | Conversion en |
|---|---|
| (int) ou (integer) | entier |
| (bool) ou (boolean) | booléen |
| (real), (double) ou (float) | nombre à virgule flottante |
| (string) | chaîne |
| (array) | tableau |
| (object) | objet |

*Exemple*

```
<?php
echo '(float)"1abc" = ',var_dump((float)"1abc"),'<br />';
echo '(float)"1.5abc" = ',var_dump((float)"1.5abc"),'<br />';
echo '(float)"abc1" = ',var_dump((float)"abc1"),'<br />';
echo '(int)1.7 = ',var_dump((int)1.7),'<br />';
echo '(int)TRUE = ',var_dump((int)TRUE),'<br />';
echo '(int)FALSE = ',var_dump((int)FALSE),'<br />';
echo '(bool)-1 = ',var_dump((bool)-1),'<br />';
echo '(bool)0 = ',var_dump((bool)0),'<br />';
echo '(bool)1 = ',var_dump((bool)1),'<br />';
echo '(bool)"" = ',var_dump((bool)""),'<br />';
echo '(bool)"0" = ',var_dump((bool)"0"),'<br />';
echo '(bool)"1" = ',var_dump((bool)"1"),'<br />';
echo '(bool)"a" = ',var_dump((bool)"a"),'<br />';
?>
```

*Résultat*

```
(float)"1abc" = float(1)
(float)"1.5abc" = float(1.5)
(float)"abc1" = float(0)
(int)1.7 = int(1)
(int)TRUE = int(1)
(int)FALSE = int(0)
(bool)-1 = bool(true)
(bool)0 = bool(false)
(bool)1 = bool(true)
(bool)"" = bool(false)
(bool)"0" = bool(false)
(bool)"1" = bool(true)
(bool)"a" = bool(true)
```

Ces différents exemples permettent de retrouver les règles de conversion évoquées dans le paragraphe précédent.

### Fonction de conversion

La fonction settype permet de convertir une variable d'un type à un autre.

*Syntaxe*

```
booléen settype(mixte variable, chaîne type)
```

variable

Variable à convertir

type

Type souhaité en utilisant une des valeurs : boolean ou bool (conversion en booléen) ; integer ou int (conversion en entier) ; double ou float (conversion en nombre à virgule flottante) ; string (conversion en chaîne de caractères) ; array (conversion en tableau) ; object (conversion en objet)

La fonction settype retourne TRUE en cas de succès et FALSE en cas d'erreur.

*Exemple*

```
<?php
$x = '1abc';
settype($x,'integer');
echo '\'1abc\' converti en entier = ',var_dump($x),'<br />';
$x = 1.7;
settype($x,'integer');
echo '1.7 converti en entier = ',var_dump($x),'<br />';
$x = TRUE;
settype($x,'string');
echo 'TRUE converti en chaîne = ',var_dump($x),'<br />';
$x = '0';
settype($x,'boolean');
echo '\'0\' converti en booléen = ',var_dump($x),'<br />';
$x = -1;
settype($x,'boolean');
echo '-1 converti en booléen = ',var_dump($x),'<br />';
?>
```

*Résultat*

```
"1abc" converti en entier = int(1)
1.7 converti en entier = int(1)
TRUE converti en chaîne = string(1) "1"
"0" converti en booléen = bool(false)
-1 converti en booléen = bool(true)
```

En règle générale, il est conseillé d'utiliser la conversion explicite : le code est plus lisible, plus facile à maintenir et à mettre au point.

### b. Fonctions utiles

En complément, PHP propose plusieurs fonctions utiles relatives au type des variables :

| **Nom** | **Rôle** |
|---|---|
| `is_*` | Indique si la variable est du type donné par *<br>`array` = tableau ;<br>`bool` = booléen ;<br>`double`, `float`, `real` = nombre à virgule flottante ;<br>`int`, `integer`, `long` = entier ;<br>`null` = type `NULL` ;<br>`numeric` = entier ou nombre à virgule flottante ou chaîne contenant un nombre (entier ou décimal) ;<br>`object` = objet ;<br>`string` = chaîne ;<br>`resource` = ressource ;<br>`scalar` = type scalaire. |
| `strval` | Convertit une variable en chaîne. |
| `floatval, doubleval` | Convertir une variable en nombre à virgule flottante. |
| `intval` | Convertir une variable en entier. |

**is_***

Chaque fonction is_* permet de tester si une variable est d'un type donné.

*Syntaxe*

```
booléen is_* (mixte variable)
```

```
variable
```

Variable à tester.

Les déclinaisons sont les suivantes :

| **Fonction** | **Type testé** |
|---|---|
| is_array | tableau |
| is_bool | booléen |
| is_double<br>is_float<br>is_real | nombre à virgule flottante |
| is_int<br>is_integer<br>is_long | Entier |
| is_null | type NULL |
| is_numeric | entier ou nombre à virgule flottante ou chaîne contenant un nombre (entier ou décimal) |
| is_object | objet |
| is_string | chaîne |
| is_resource | ressource |
| is_scalar | type scalaire |

Ces fonctions retournent TRUE si la variable est du type demandé et FALSE dans le cas contraire.

*Exemple*

```
<?php
if (is_null($x)) {
  echo 'Pour l\'instant, $x est du type NULL.<br />';
}
$x = (1 < 2);
if (is_bool($x)) {
  echo '$x = (1 < 2) est du type booléen.<br />';
}
$x = '123abc';
if (is_string($x)) {
  echo '$x = "123abc" est du type chaîne ...<br />';
}
if (! is_numeric($x)) {
  echo '... mais pas du « type » <i>numeric</i>.<br />';
}
?>
```

*Résultat*

```
Pour l'instant, $x est du type NULL.
$x = (1 < 2) est du type booléen.
$x = "123abc" est du type chaîne ...
... mais pas du « type » numeric.
```

Cet exemple montre que la fonction is_numeric n'applique pas tout à fait les mêmes règles pour évaluer si une chaîne contient un nombre que celles utilisées pour la conversion. Avec la fonction is_numeric, la chaîne ne doit contenir aucun caractère non numérique.

### strval

La fonction strval retourne la valeur d'une variable après conversion en chaîne.

*Syntaxe*

```
chaîne strval(mixte variable)
```

`variable`

Variable à traiter.

Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets). Le type de la variable est inchangé.

*Exemple*

```
<?php
$x = TRUE;
echo var_dump($x),' => ',var_dump(strval($x)),'<br />';
$x = 1.2345;
echo var_dump($x),' => ',var_dump(strval($x)),'<br />';
?>
```

*Résultat*

```
bool(true) => string(1) "1"
float(1.2345) => string(6) "1.2345"
```

## floatval (ou doubleval)

La fonction `floatval` retourne la valeur d'une variable après conversion en nombre à virgule flottante. La fonction `doubleval` est un alias de la fonction `floatval`.

*Syntaxe*

```
nombre doubleval(mixte variable)
```

`variable`

Variable à traiter.

Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets). Le type de la variable est inchangé. Les règles de conversions évoquées précédemment sont respectées.

*Exemple*

```
<?php
$x = TRUE;
echo var_dump($x)," => ",var_dump(doubleval($x)),'<br />';
$x = 123;
echo var_dump($x)," => ",var_dump(doubleval($x)),'<br />';
$x = "1.2345";
echo var_dump($x)," => ",var_dump(doubleval($x)),'<br />';
?>
```

*Résultat*

```
bool(true) => float(1)
int(123) => float(123)
string(6) "1.2345" => float(1.2345)
```

## intval

La fonction `intval` retourne la valeur d'une variable après conversion en entier.

*Syntaxe*

```
nombre intval(mixte variable)
```

`variable`

Variable à traiter.

Cette fonction ne s'applique qu'aux variables de type scalaire (non valable pour les tableaux, ni les objets). Le type de la variable est inchangé. Les règles de conversions évoquées précédemment sont respectées.

*Exemple*

```
<?php
$x = TRUE;
echo var_dump($x),' => ',var_dump(intval($x)),'<br />';
$x = 123.9;
echo var_dump($x),' => ',var_dump(intval($x)),'<br />';
$x = "9.99";
echo var_dump($x),' => ',var_dump(intval($x)),'<br />';
?>
```

*Résultat*

```
bool(true) => int(1)
float(123.9) => int(123)
string(4) "9.99" => int(9)
```

N'oubliez pas qu'un nombre à virgule flottante converti en entier est tronqué et non arrondi : sur notre exemple, `123.9` donne `123` et non `124`. Pour convertir un nombre à virgule flottante en entier, avec arrondi, il faut utiliser la fonction `round()`.

*Exemple*

```
<?php
$x = 123.9;
echo "round($x) => ";
var_dump(round($x));
?>
```

*Résultat*

```
round(123.9) => float(124)
```

La fonction `round()` retourne bien un nombre entier arrondi, mais avec un type de nombre à virgule flottante. Pour obtenir une donnée de type entier, il suffit de convertir le résultat de `round()` en entier (avec la fonction `intval` ou la construction `(int)`).

# Manipuler les tableaux

PHP propose un très grand nombre de fonctions permettant de manipuler les tableaux.

Les fonctions les plus utilisées sont les suivantes :

| Nom | Rôle |
|---|---|
| count | Compte le nombre d'éléments dans un tableau. |
| in_array | Teste si une valeur est présente dans un tableau. |
| array_search | Recherche une valeur dans un tableau. |
| [a\|k][r]sort | Trient un tableau (plusieurs variantes possibles). |
| explode | Découpe une chaîne selon un séparateur et stocke les éléments dans un tableau. |
| str_split | Découpe une chaîne en morceaux de longueur fixe et stocke les éléments dans un tableau. |
| implode | Regroupe les éléments d'un tableau dans une chaîne à l'aide d'un séparateur. |

N'oubliez pas la fonction is_array qui permet de savoir si une variable est de type tableau (cf. dans ce chapitre - Manipuler les constantes, les variables et les types de données - Types de données).

De nombreuses autres fonctions existent. La description de chaque fonction est accessible en ligne sur le site **www.php.net**. Vous y trouverez notamment des fonctions pour :

- réaliser des calculs (somme, ...) ;
- extraire un sous-tableau d'un tableau ;
- fusionner des tableaux ;
- dédoublonner un tableau, etc.

## count

La fonction count permet de connaître le nombre d'éléments dans une variable en général, un tableau en particulier.

*Syntaxe*

```
entier count (mixte variable)
```

variable

Variable concernée

La fonction count retourne le nombre d'éléments dans la variable. Si la variable n'est pas initialisée, count retourne 0. Si la variable est initialisée mais n'est pas un tableau, count retourne 1 (il y a effectivement un élément dans une variable scalaire). Si la variable est un tableau, count retourne le nombre d'éléments présents dans le tableau (0 si le tableau est vide).

*Exemple*

```
<?php
echo '$x non initialisé => ',count($x),'<br />';
$x = 1;
```

```
echo '$x de type scalaire => ',count($x),'<br />';
$x = array();
echo '$x tableau vide => ',count($x),'<br />';
$x = array(1,2);
echo '$x tableau de 2 éléments => ',count($x),'<br />';
?>
```

*Résultat*

```
$x non initialisé => 0
$x de type scalaire => 1
$x tableau vide => 0
$x tableau de 2 éléments => 2
```

## in_array

La fonction in_array permet de tester si une valeur est présente dans un tableau.

*Syntaxe*

```
booléen in_array(mixte valeur_cherchée, tableau tableau[, booléen
même_type])
```

valeur_cherchée

Valeur cherchée dans le tableau.

tableau

Tableau dans lequel s'effectue la recherche.

même_type

Indique si la comparaison doit vérifier que les éléments sont du même type (FALSE par défaut).

La fonction in_array retourne TRUE si l'élément cherché est présent dans le tableau et FALSE dans le cas contraire.


PHP propose aussi la fonction array_key_exists qui permet de tester si une valeur est présente dans les clés d'un tableau.

*Exemple*

```
<?php
$nombres = array('zéro','un','deux',
                 'zéro' => 0,'un' => 1,'deux' => 2);
echo '1 type indifférent => ',
     var_dump(in_array(1,$nombres)),'<br />';
echo '3 type indifférent => ',
     var_dump(in_array(3,$nombres)),'<br />';
echo '\'1\' même type => ',
     var_dump(in_array('1',$nombres,TRUE)),'<br />';
echo '\'un\' type indifférent => ',
     var_dump(in_array('un',$nombres)),'<br />';
echo '\'trois\' type indifférent => <b>',
     var_dump(in_array('trois',$nombres)),'</b><br />';
echo '\'trois\' même type => ',
     var_dump(in_array('trois',$nombres,TRUE)),'<br />'
;?>
```

*Résultat*

```
1 type indifférent => bool(true)
3 type indifférent => bool(false)
'1' même type => bool(false)
'un' type indifférent => bool(true)
'trois' type indifférent => bool(true)
'trois' même type => bool(false)
```

Les deux exemples avec 'trois' montrent que la fonction in_array est à manipuler avec beaucoup de précaution lorsque le tableau contient des éléments de type différent. En effet, sur la recherche de même type, la fonction fonctionne correctement (il n'y a pas de chaîne 'trois' dans le tableau). Sur la recherche où le type est indifférent, tout se passe comme si la chaîne était convertie en entier ('trois' converti en entier donne 0) et que la recherche s'effectuait sur le résultat de cette conversion (0 est bien présent dans le tableau).

## array_search

La fonction array_search permet de chercher un élément dans un tableau et de récupérer la clé de cet élément, s'il est présent.

*Syntaxe*

```
mixte array_search(mixte valeur_cherchée, tableau tableau[, booléen
même_type])
```

valeur_cherchée

Valeur cherchée dans le tableau.

tableau

Tableau dans lequel s'effectue la recherche.

même_type

Indique si la comparaison doit vérifier que les éléments sont du même type (FALSE par défaut).

La fonction array_search retourne la clé associée à l'élément si ce dernier est présent dans le tableau et FALSE dans le cas contraire (NULL avant la version 4.2.0).

*Exemple*

```
<?php
$nombres = array('zéro','un','deux',
                 'zéro' => 0,'un' => 1,'deux' => 2);
echo '1 type indifférent => ',
     var_dump(array_search(1,$nombres)),'<br />';
echo '3 type indifférent => ',
     var_dump(array_search(3,$nombres)),'<br />';
echo '\'1\' même type => ',
     var_dump(array_search('1',$nombres,TRUE)),'<br />';
echo '\'un\' type indifférent => ',
     var_dump(array_search('un',$nombres)),'<br />';
echo '\'trois\' type indifférent => <b>',
     var_dump(array_search('trois',$nombres)),'</b><br />';
echo '\'trois\' même type => ',
     var_dump(array_search('trois',$nombres,TRUE)),'<br />'
;?>
```

*Résultat*

```
1 type indifférent => string(2) "un"
3 type indifférent => bool(false)
'1' même type => bool(false)
'un' type indifférent => int(1)
'trois' type indifférent => string(4) "zéro"
'trois' même type => bool(false)
```

Nous retrouvons sur ces exemples le problème évoqué avec la fonction in_array : la recherche sur 'trois', en type indifférent, donne la clé 'zéro' qui correspond effectivement à la valeur 0 dans le tableau.

## [a|k][r]sort

Les fonctions sort, rsort, asort, arsort, ksort et krsort permettent de trier un tableau selon différentes variantes.

*Syntaxe*

`booléen [a|k][r]sort(tableau tableau, entier indicateur)`

`tableau`

Tableau à trier.

`indicateur`

Paramètre optionnel permettant de modifier le comportement du tri :

- `SORT_REGULAR` (valeur par défaut) : compare les éléments normalement (ne modifie pas les types).
- `SORT_NUMERIC` : compare les éléments numériquement.
- `SORT_STRING` : compare les éléments comme des chaînes de caractères.
- `SORT_LOCALE_STRING` (apparu dans la version 5.0.2) : compare les éléments en utilisant la configuration locale définie par la fonction `setlocale`.

---

Trier un tableau comportant des valeurs de différents types donne un résultat imprévisible.

---

Les variantes de fonctionnement sont les suivantes :

| **Fonction** | **Nature du tri** |
|---|---|
| `sort` | Tri croissant sur la valeur, sans préservation des couples clé/valeur.<br>Quelle que soit la situation de départ, après le tri, les indices du tableau sont des entiers consécutifs à partir de 0. |
| `rsort` | Tri décroissant sur la valeur, sans préservation des couples clé/valeur.<br>Quelle que soit la situation de départ, après le tri, les indices du tableau sont des entiers consécutifs à partir de 0. |
| `asort` | Tri croissant sur la valeur, avec préservation des couples clé/valeur. |
| `arsort` | Tri décroissant sur la valeur, avec préservation des couples clé/valeur. |
| `ksort` | Tri croissant sur la clé, avec préservation des couples clé/valeur. |
| `krsort` | Tri décroissant sur la clé, avec préservation des couples clé/valeur. |

Ces fonctions retournent `TRUE` en cas de succès et `FALSE` en cas d'échec.

*Exemple*

```
<?php
$tableau = array('c3' => 'rouge','c1' => 'vert',
                 'c2' => 'bleu');
// Affichage de contrôle.
echo '<b>Tableau de départ :</b><br />';
foreach($tableau as $clé => $valeur)
  { echo "$clé => $valeur<br />"; }
// sort
echo '<b>sort : tri sur valeur, clés non préservées</b><br />';
```

```
$tableau_bis = $tableau;
sort($tableau_bis);
foreach($tableau_bis as $clé => $valeur)
  { echo "$clé => $valeur<br />"; }
// asort
echo '<b>asort : tri sur valeur, clés préservées</b><br />';
$tableau_bis = $tableau;
asort($tableau_bis);
foreach($tableau_bis as $clé => $valeur)
  { echo "$clé => $valeur<br />"; }
// ksort
echo '<b>ksort : tri sur clé, clés préservées</b><br />';
$tableau_bis = $tableau;
ksort($tableau_bis);
foreach($tableau_bis as $clé => $valeur)
  { echo "$clé => $valeur<br />"; }
?>
```

*Résultat*

```
Tableau de départ :
c3 => rouge
c1 => vert
c2 => bleu
sort : tri sur valeur, clés non préservées
0 => bleu
1 => rouge
2 => vert
asort : tri sur valeur, clés préservées
c2 => bleu
c3 => rouge
c1 => vert
ksort : tri sur clé, clés préservées
c1 => vert
c2 => bleu
c3 => rouge
```

### explode

La fonction `explode` permet de découper une chaîne selon un séparateur et de stocker les éléments dans un tableau.

*Syntaxe*

```
tableau explode(chaîne séparateur, chaîne à_découper[, entier limite])
```

`séparateur`

Séparateur recherché.

`à_découper`

Chaîne à découper.

`limite`

Si spécifié, nombre maximum d'éléments dans le tableau résultat.

*Exemple*

```
<?php
$liste = 'bleu, blanc, rouge';
$couleurs = explode(', ',$liste);
  // séparateur = virgule+espace
foreach($couleurs as $clé => $valeur)
  { echo "$clé => $valeur<br />"; }
?>
```

*Résultat*

```
0 => bleu
1 => blanc
2 => rouge
```

## str_split

La fonction `str_split` découpe une chaîne en morceaux de longueur fixe et stocke les éléments dans un tableau.

*Syntaxe*

```
tableau str_split(chaîne chaîne[,entier longueur])
```

`chaîne`

Chaîne à découper.

`longueur`

Longueur des morceaux (1 par défaut).

*Exemple*

```
<?php
$chaîne = 'A1B2C3';
$tableau = str_split($chaîne,2);
foreach($tableau as $clé => $valeur) {
  echo "\$tableau[$clé] = $valeur<br>";
}
?>
```

*Résultat*

```
$tableau[0] = A1
$tableau[1] = B2
$tableau[2] = C3
```

## implode

La fonction `implode` permet de regrouper les éléments d'un tableau dans une chaîne à l'aide d'un séparateur

```
chaîne implode(chaîne séparateur, tableau éléments)
```

`séparateur`

Séparateur utilisé.

`éléments`

Tableau contenant les éléments à regrouper.

*Exemple*

```
<?php
$couleurs = array('bleu','blanc','rouge');
$liste = implode(', ',$couleurs);
  // séparateur = virgule+espace
echo $liste;
?>
```

*Résultat*

```
bleu, blanc, rouge
```

# Manipuler les chaînes de caractères

Les fonctions les plus utiles pour manipuler les chaînes de caractères sont les suivantes :

| Nom | Rôle |
|---|---|
| strlen | Retourne le nombre de caractères d'une chaîne. |
| strtolower<br>strtoupper<br>ucfirst<br>ucwords | Conversions minuscules/majuscules. |
| strcmp<br>strcasecmp | Comparaison de chaînes (sensible à la casse ou non). |
| [s]printf<br>v[s]printf | Mise en forme d'une chaîne (identique aux fonctions C équivalentes). |
| number_format | Mise en forme d'un nombre. |
| [l\|r]trim | Suppression de caractères "blancs". |
| substr | Extraction d'une portion de chaîne. |
| str_repeat | Construction d'une chaîne par répétition de caractères. |
| str[r][i]pos | Recherche la position d'une occurrence (caractère ou chaîne) à l'intérieur d'une chaîne. |
| str[i]str<br>strrchr | Extraction de la sous-chaîne commençant à partir d'une certaine occurrence d'un caractère ou d'une chaîne. |
| str_[i]replace | Remplacement des occurrences d'une chaîne par une autre chaîne. |
| strtr | Remplacement des occurrences d'un caractère par un autre caractère ou d'une chaîne par une autre chaîne. |
| ereg[i]<br>ereg[i]_replace | Recherche et remplacement à l'aide d'expressions régulières. |

N'oubliez pas les fonctions `explode` et `implode` présentées précédemment (cf. dans ce chapitre - Manipuler les tableaux).

D'autres fonctions, plus spécifiquement liées à la gestion des "guillemets magiques" sont étudiées plus loin dans ce chapitre (cf. dans ce chapitre - Gérer les "guillemets magiques" ("magic quotes")).

## strlen

La fonction `strlen` retourne le nombre de caractères d'une chaîne.

*Syntaxe*

```
entier strlen(chaîne chaîne)
```

`chaîne`

Chaîne concernée

*Exemple*

```
<?php
$x = 'Olivier Heurtel';
echo "strlen('$x') = ",strlen($x);
?>
```

*Résultat*

```
strlen('Olivier Heurtel') = 15
```

## strtolower - strtoupper - ucfirst - ucwords

Ces fonctions permettent de faire des conversions minuscules/majuscules.

*Syntaxe*

```
chaîne strtolower(chaîne chaîne)
chaîne strtoupper(chaîne chaîne)
chaîne ucfirst(chaîne chaîne)
chaîne ucwords(chaîne chaîne)
```

`chaîne`

Chaîne à traiter

La fonction `strtolower` convertit tous les caractères d'une chaîne en minuscules.

La fonction `strtoupper` convertit tous les caractères d'une chaîne en majuscules.

La fonction `ucfirst` convertit le premier caractère d'une chaîne en majuscules.

La fonction `ucwords` convertit le premier caractère de chaque mot d'une chaîne en majuscule.

*Exemple*

```
<?php
$x = 'OLIVIER HEURTEL';
$y = 'olivier heurtel';
echo "strtolower('$x') = ",strtolower($x),'<br />';
echo "strtoupper('$y') = ",strtoupper($y),'<br />';
echo "ucfirst('$y') = ",ucfirst($y),'<br />';
echo "ucwords('$y') = ",ucwords($y),'<br />';
?>
```

*Résultat*

```
strtolower('OLIVIER HEURTEL') = olivier heurtel
strtoupper('olivier heurtel') = OLIVIER HEURTEL
ucfirst('olivier heurtel') = Olivier heurtel
ucwords('olivier heurtel') = Olivier Heurtel
```

## strcmp - strcasecmp

Ces fonctions permettent de comparer deux chaînes en tenant compte ou pas des majuscules et minuscules.

*Syntaxe*

```
entier strcmp(chaîne chaîne1,chaîne chaîne2)
entier strcasecmp(chaîne chaîne1,chaîne chaîne2)
```

`chaîne1` et `chaîne2`

Chaînes à comparer.

Les deux fonctions retournent un nombre négatif si `chaîne1` est plus petit que `chaîne2`, un nombre égal à 0 si elles sont égales, et un nombre positif si `chaîne1` est plus grand que `chaîne2`.

`strcmp` est sensible à la casse alors que `strcasecmp` ne l'est pas.

*Exemple*

```
<?php
$x = 'Olivier';
$y = 'olivier';
echo "strcmp('$x','$y') = ",strcmp($x,$y),'<br />';
echo "strcasecmp('$x','$y') = ",strcasecmp($x,$y),'<br />';
?>
```

*Résultat*

```
strcmp('Olivier','olivier') = -1
strcasecmp('Olivier','olivier') = 0
```

**[s]printf**

Les fonctions `printf` et `sprintf` permettent de mettre en forme une chaîne (identiques aux fonctions C équivalentes).

*Syntaxe*

```
chaîne sprintf(chaîne format[, mixte valeur[, ...]])
chaîne printf(chaîne format[, mixte valeur[, ...]])
```

`format`

Chaîne de mise en forme présentant diverses directives selon les spécifications données ci-après.

`valeur`

Valeur à intégrer dans la chaîne.

La fonction `sprintf` retourne le résultat mis en forme (ou `FALSE` en cas d'erreur) alors que `printf` affiche directement le résultat (comme l'instruction `echo`) et retourne la longueur de la chaîne mise en forme en cas de succès ou `FALSE` en cas d'erreur.

La chaîne `format` doit contenir une directive de formatage pour chaque argument `valeur` ; cette directive de formatage précise l'emplacement et la mise en forme de la valeur correspondante. La correspondance entre une directive de formatage et une valeur est positionnelle (première directive pour la première valeur...).

Les directives de formatage commencent par le caractère `%` suivi d'une à cinq informations, la dernière étant la seule obligatoire :

```
%[remplissage][alignement][longueur][précision]type
```

Les informations sont les suivantes :

`remplissage`

Précise le caractère éventuel utilisé pour le remplissage. Le caractère par défaut est l'espace. Tout autre caractère peut être utilisé en le mentionnant précédé d'une apostrophe (seul le caractère zéro peut être indiqué directement) : `'x` indique que le caractère de remplissage est le "x".

`alignement`

Précise l'alignement. Par défaut, l'alignement est à droite. Le caractère moins ("-") permet d'obtenir un alignement à gauche.

`longueur`

Indique le nombre minimum de caractères de l'élément mis en forme.

`précision`

Indique le nombre de chiffres utilisés pour la mise en forme d'un nombre à virgule flottante (valable uniquement si l'élément associé est un nombre).

`type`

Donne le type de la valeur à insérer : `c` : entier à remplacer par le caractère dont le code ASCII a cette valeur ; `d` : entier à représenter comme tel ; `f` : nombre à virgule flottante à représenter comme tel (tient compte de la configuration locale utilisée) ; `F` : nombre à virgule flottante à représenter comme tel (ne tient pas compte de la configuration locale utilisée) ; `s` : quelconque, à représenter comme une chaîne.

Pour avoir un caractère "%" dans le résultat final, il faut le doubler dans le format.

Quelques exemples :

| Directive | Valeur | Résultat | Explication |
|---|---|---|---|

| | | | |
|---|---|---|---|
| %d | 1 | 1 | Nombre entier sans mise en forme particulière. |
| %02d | 1 | 01 | 02 = compléter avec le caractère zéro, pour une longueur de deux minimum. |
| %f | 1/3 | 0.333333 | Nombre à virgule flottante sans mise en forme particulière. |
| %.2f | 1/3 | 0.33 | .2 = deux chiffres après le séparateur décimal. |
| %s | Olivier | Olivier | Chaîne sans mise en forme particulière. |
| %'.-10s | Olivier | Olivier... | '.-10 = compléter avec un point pour atteindre une longueur minimum de dix caractères (signe - = alignement à gauche) |
| %'.5.2f | 9.9 | ....9.90 | .2 = deux chiffres après le séparateur décimal. '.5 = compléter avec un point pour atteindre une longueur minimum de 5 caractères avant le point décimal (alignement par défaut) |

*Exemple*

```
<?php
echo 'Mise en forme d'une date : ',
     sprintf('%02d/%02d/%04d',1,1,2001),'<br />';
echo 'Mise en forme de nombres : ',
     sprintf('%01.2f - %01.2f',1/3,12345678.9),'<br />';
echo 'Pourcentage : ',
     sprintf('%01.2f %%',12.3),'<br />';
echo 'Utilisation des options de remplissage :<br />';
echo '<tt>'; // police non proportionnelle
printf("%'.-10s%'.5.2f<br />",'Livres',9.35); // printf direct
printf("%'.-10s%'.5.2f<br />",'Disques',99.9); // printf direct
echo '</tt>';
?>
```

*Résultat*

```
Mise en forme d'une date : 01/01/2001
Mise en forme de nombres : 0.33 - 12345678.90
Pourcentage : 12.30 %
Utilisation des options de remplissage :
Livres.....9.35
Disques...99.90
```

### v[s]printf

Les fonctions vprintf et vsprintf sont identiques aux fonctions printf et sprintf mais acceptent en deuxième paramètre un tableau contenant les différentes valeurs à utiliser (à la place des paramètres multiples).

*Syntaxe*

```
chaîne vsprintf(chaîne format[, tableau valeurs])
chaîne vprintf(chaîne format[, tableau valeurs])
```

format

Chaîne de mise en forme comportant diverses directives selon les spécifications données précédemment.

`valeurs`

Tableau donnant les valeurs à intégrer dans la chaîne.

*Exemple*

```
<?php
$données = array(array('Livres',9.35),array('Disques',99.9));
echo '<tt>'; // police non proportionnelle
foreach($données as $ligne) {
  vprintf("%'.-10s%'.5.2f<BR>",$ligne); // printf direct
}
echo "</tt>";
?>
```

*Résultat*

```
Livres.....9.35
Disques...99.90
```

## number_format

La fonction `number_format` permet de mettre en forme un nombre.

*Syntaxe*

```
chaîne number_format(nombre valeur[, entier décimales[,
chaîne séparateur_décimal, chaîne séparateur_milliers]])
```

`valeur`

Nombre à mettre en forme.

`décimales`

Nombre de décimales (aucune partie décimale par défaut).

`séparateur_décimal`

Séparateur décimal (point par défaut).

`séparateur_milliers`

Séparateur des milliers (virgule par défaut).

La fonction peut être appelée avec un, deux ou quatre arguments, pas trois : si le troisième est fourni, le quatrième est obligatoire.

Si le nombre a une précision supérieure à celle demandée (paramètre `décimales`), le nombre est arrondi à la précision demandée.

*Exemple*

```
<?php
$x = 1234.567;
echo "number_format($x) = ",number_format($x),'<br />';
echo "number_format($x,1,',',' ') = ",
      number_format($x,2,',',' '),'<br />';
?>
```

*Résultat*

```
number_format(1234.567) = 1,235
number_format(1234.567,1,',',' ') = 1 234,57
```

Notez sur cet exemple que le nombre a été arrondi à la précision demandée.

## ltrim - rtrim - trim

Ces fonctions permettent de supprimer les caractères "blancs", ou d'autres caractères, en début de chaîne, en fin de chaîne ou

des deux côtés.

*Syntaxe*

```
chaîne ltrim(chaîne chaîne[, chaîne caractères])
chaîne rtrim(chaîne chaîne[, chaîne caractères])
chaîne trim(chaîne chaîne[, chaîne caractères])
```

`chaîne`

Chaîne à traiter.

`caractères`

Chaîne donnant la liste des caractères à supprimer. Si ce paramètre est absent, les caractères "blancs" sont supprimés.

Les trois fonctions retournent une chaîne égale à la chaîne initiale dans laquelle les caractères "blancs" ou les caractères spécifiés ont été supprimés au début (`ltrim`, `l` = *left* = à gauche), à la fin (`rtrim`, `r` = *right* = à droite) ou des deux côtés (`trim`).

Les caractères "blancs" sont le saut de ligne (`\n` = code ASCII 10), le retour chariot (`\r` = code ASCII 13), la tabulation (`\t` = code ASCII 9), le caractère `NUL` (`\0` = code ASCII 0) et l'espace.

*Exemple*

```
<?php
$x = "\t\t\t x \n\r";
echo 'strlen($x) = ',strlen($x),'<br />';
echo 'strlen(ltrim($x)) = ',strlen(ltrim($x)),'<br />';
echo 'strlen(rtrim($x)) = ',strlen(rtrim($x)),'<br />';
echo 'strlen(trim($x)) = ',strlen(trim($x)),'<br />';
$x = '***+-Olivier-+***';
echo "trim('$x','*+-') = ",trim($x,'*+-'), '<br />';
?>
```

*Résultat*

```
strlen($x) = 8
strlen(ltrim($x)) = 4
strlen(rtrim($x)) = 5
strlen(trim($x)) = 1
trim('***+-Olivier-+***','*+-') = Olivier
```

### substr

La fonction `substr` permet d'extraire une portion d'une chaîne.

*Syntaxe*

```
chaîne substr(chaîne chaîne, entier début[, entier longueur])
```

`chaîne`

Chaîne à traiter.

`début`

Position du premier caractère de la sous-chaîne à extraire (attention : 0 = 1er caractère)

`longueur`

Nombre de caractères à extraire (par défaut, jusqu'à la fin de la chaîne).

- Si `début` est positif, la sous-chaîne extraite commence au caractère `début` (0 = 1er caractère).
- Si `début` est négatif, la sous-chaîne extraite commence au caractère `début` en partant de la fin(-1 = dernier caractère).
- Si `longueur` n'est pas spécifié, la sous-chaîne extraite se termine à la fin de la chaîne.
- Si `longueur` est spécifié et positif, `substr` extrait le nombre de caractères indiqué par `longueur`.

- Si longueur est spécifié et négatif, la portion extraite se termine à la fin de la chaîne moins le nombre de caractères indiqué par la valeur absolue de longueur.

*Exemple*

```
<?php
//    0123456 => pour le contrôle
$x = 'Olivier';
echo "substr('$x',3) = ",substr($x,3),'<br />';
echo "substr('$x',3,2) = ",substr($x,3,2),'<br />';
echo "substr('$x',-4) = ",substr($x,-4),'<br />';
echo "substr('$x',-4,3) = ",substr($x,-4,3),'<br />';?>
```

*Résultat*

```
substr('Olivier',3) = vier
substr('Olivier',3,2) = vi
substr('Olivier',-4) = vier
substr('Olivier',-4,3) = vie
```

## str_repeat

La fonction str_repeat permet de construire une chaîne par répétition de caractères.

*Syntaxe*

```
chaîne str_repeat(chaîne séquence, entier répétitions)
```

séquence

Séquence de caractères à répéter.

répétitions

Nombre de répétitions souhaitées.

*Exemple*

```
<?php
echo str_repeat('abc',3);
?>
```

*Résultat*

```
abcabcabc
```

## strpos - strrpos - stripos - strripos

Ces fonctions permettent de rechercher la position d'une occurrence (caractère ou chaîne) à l'intérieur d'une chaîne.

*Syntaxe*

```
entier strpos(chaîne à_traiter, chaîne chercher[, entier début])
entier strrpos (chaîne à_traiter, chaîne chercher[, entier début])
entier stripos(chaîne à_traiter, chaîne chercher[, entier début])
entier strripos (chaîne à_traiter, chaîne chercher[, entier début])
```

à_traiter

Chaîne à traiter.

chercher

Élément recherché.

début

Numéro du caractère (0 = premier caractère) à partir duquel la recherche doit être réalisée (par défaut, le début de la chaîne).

La fonction strpos recherche, dans la chaîne à_traiter, la première occurrence de la chaîne chercher, en commençant éventuellement au caractère numéro début (0 = premier caractère).

La fonction strrpos recherche, dans la chaîne à_traiter, la dernière occurrence de la chaîne chercher, en commençant éventuellement au caractère numéro début (0 = premier caractère) Si début est négatif (-n), les n derniers caractères de la chaîne à_traiter sont ignorés.

Les deux fonctions sont sensibles à la casse (une majuscule n'est pas égale à une minuscule). Les fonctions stripos et strripos sont identiques respectivement aux fonctions strpos et strrpos, mais elles ne sont pas sensibles à la casse.

Ces quatre fonctions retournent la position de l'occurrence trouvée (0 = premier caractère) ou FALSE si l'élément recherché n'est pas trouvé.

FALSE étant équivalent à 0, il est facile de confondre le cas où l'élément n'a pas été trouvé et le cas où il a été trouvé en début de chaîne. La technique consiste à utiliser l'opérateur de comparaison === (trois signes égal) qui permet de comparer la valeur et le type de deux expressions (pour plus de détail, cf. chapitre Introduction à PHP - Les base du langage PHP).

*Exemple*

```
<?php
//       0123456789 ... => pour le contrôle
$mail = 'contact@olivier-heurtel.fr';
// strrpos
$position = strrpos($mail,'@');
echo "@ est à la position $position dans $mail<br />";
// strpos
$position = strpos($mail,'Olivier');
echo "'Olivier' est à la position $position dans $mail<br />";
// occurrence en début de chaîne
$position = strpos($mail,'contact');
if ($position === FALSE) { // bon test : ===
  echo "'contact' est introuvable dans $mail<br />";
} else {
  echo "'contact' est à la position $position
          dans $mail<br />";
}
// occurrence non trouvée
$position = strpos($mail,'information');
if ($position === FALSE) { // bon test : ===
  echo "'information' est introuvable dans $mail<br />";
} else {
  echo "'information' est à la position $position
          dans $mail<br />";
}
?>
```

*Résultat*

```
@ est à la position 7 dans contact@olivier-heurtel.fr
'Olivier' est à la position dans contact@olivier-heurtel.fr
'contact' est à la position 0 dans contact@olivier-heurtel.fr
'information' est introuvable dans contact@olivier-heurtel.fr
```

### strstr - stristr - strrchr

Ces fonctions permettent d'extraire la sous-chaîne commençant à partir d'une certaine occurrence d'un caractère ou d'une chaîne.

*Syntaxe*

```
chaîne strstr(chaîne à_traiter, chaîne rechercher)
chaîne stristr(chaîne à_traiter, chaîne rechercher)
chaîne strrchr(chaîne à_traiter, caractère rechercher)
```

à_traiter

Chaîne à traiter.

chercher

Élément recherché.

Les fonctions strstr et stristr recherchent, dans la chaîne à_traiter, la première occurrence de la chaînerechercher, et retournent la portion terminale de la chaîne commençant à cette occurrence (incluse). La fonction strstr est sensible à la casse (une majuscule est différente d'une minuscule) alors que stristr ne l'est pas.

La fonction strrchr recherche, dans la chaîne à_traiter, la dernière occurrence du caractèrerechercher et retourne la portion

terminale de la chaîne commençant à cette occurrence (incluse). Si `rechercher` est une chaîne de plusieurs caractères, seul le premier est pris en compte. La fonction `strrchr` est sensible à la casse.

Ces trois fonctions retournent FALSE si l'élément recherché n'est pas trouvé.

*Exemple*

```
<?php
$mail = 'Olivier-Heurtel@olivier-heurtel.fr';
echo "Reste de $mail commençant par :<br />";
// strrchr
$reste = strrchr($mail,'-');
echo "- la dernière occurrence de '-'<br />----> $reste <br />";
// strstr
$reste = strstr($mail,'olivier');
echo "- la première occurrence de 'olivier'
      (sensible à la casse)<br />----> $reste <br />";
// stristr
$reste = stristr($mail,'olivier');
echo "- la première occurrence de 'olivier'
      (insensible à la casse)<br />----> $reste <br />";
?>
```

*Résultat*

```
Reste de Olivier-Heurtel@olivier-heurtel.fr commençant par :
- la dernière occurrence de '-'
----> -heurtel.fr
- la première occurrence de 'olivier' (sensible à la casse)
----> olivier-heurtel.fr
- la première occurrence de 'olivier' (insensible à la casse)
----> Olivier-Heurtel@olivier-heurtel.fr
```

## str_replace - str_ireplace

La fonction `str_replace` permet de remplacer les occurrences d'une chaîne par une autre chaîne. La recherche est sensible à la casse.

La fonction `str_ireplace` permet la même action mais n'est pas sensible à la casse.

*Syntaxe*

```
mixte str_replace(mixte rechercher, mixte remplacer, mixte à_traiter[,
entier nombre])mixte str_ireplace(mixte rechercher, mixte remplacer,
mixte à_traiter[, entier nombre])
```

`à_traiter`

Chaîne à traiter ou tableau de chaînes à traiter.

`rechercher`

Chaîne à remplacer ou tableau donnant une liste de chaînes à rechercher.

`remplacer`

Chaîne de remplacement ou tableau donnant une liste de chaînes de remplacement.

`nombre`

Variable permettant de récupérer le nombre de remplacements.

Si `rechercher` et `remplacer` sont des chaînes, la fonction `str_replace` recherche toutes les occurrences de `rechercher` et les remplace par `remplacer`.

Si `rechercher` et `remplacer` sont des tableaux, la fonction `str_replace` recherche toutes les occurrences de chaque élément de `rechercher` et les remplace par l'élément correspondant de `remplacer`.

Dans les deux cas, le traitement est effectué sur la chaîne `à_traiter` ou sur chaque élément de `à_traiter` si cette dernière est un tableau.

*Exemple*

```
<?php
```

```
// première syntaxe
$x = 'cet été, à la plage';
$rechercher = 'été';
$remplacer = 'hiver';
echo '<b>Première syntaxe :</b><br />';
echo "$rechercher => $remplacer <br />";
echo "$x => ",str_replace($rechercher,$remplacer,$x),'<br />';
// deuxième syntaxe
$x = array('cet été, à la plage','le bateau bleu et vert');
$rechercher = array('été','plage','bleu','vert');
$remplacer = array('hiver','montagne','rouge','jaune');
echo "<b>Deuxième syntaxe :</b><br />";
foreach($rechercher as $indice => $avant)
  { echo "$avant => $remplacer[$indice]<br />"; }
// Utilisation de la variable $nombre pour récupérer
// le nombre de remplacements.
$y = str_replace($rechercher,$remplacer,$x,$nombre);
echo "$x[0] => $y[0]<br />";
echo "$x[1] => $y[1]<br />";
echo "$nombre remplacements<br />";
?>
```

*Résultat*

```
Première syntaxe :
été => hiver
cet été, à la plage => cet hiver, à la plage
Deuxième syntaxe :
été => hiver
plage => montagne
bleu => rouge
vert => jaune
cet été, à la plage => cet hiver, à la montagne
le bateau bleu et vert => le bateau rouge et jaune
4 remplacements
```

## strtr

La fonction `strtr` permet de remplacer les occurrences d'un caractère par un autre caractère ou d'une chaîne par une autre chaîne

*Syntaxe*

```
chaîne strtr(chaîne à_traiter, chaîne rechercher, chaîne remplacer)
ou
chaîne strtr(chaîne à_traiter, tableau correspondance)
```

`à_traiter`

Chaîne à traiter.

`rechercher`

Chaîne donnant la liste des caractères à remplacer.

`remplacer`

Chaîne donnant la liste des caractères de remplacement.

`correspondance`

Tableau associatif donnant une correspondance chaîne à remplacer / chaîne de remplacement

La fonction `strtr` accepte deux syntaxes : la première permet de remplacer des caractères par d'autres et la deuxième de remplacer des chaînes par d'autres.

Avec la première syntaxe, la correspondance entre les caractères à remplacer et les caractères de remplacement est donnée par deux chaînes (le caractère n de la première étant à remplacer par le caractère n de la deuxième).

Avec la deuxième syntaxe, la correspondance entre les chaînes à remplacer et les chaînes de remplacement est donnée par un tableau associatif (la clé étant la chaîne à rechercher et la valeur la chaîne de remplacement).

*Exemple*

```
<?php
// première syntaxe
$x = 'cet été, à la plage';
$avant = 'éèà';
$après = 'eea';
echo '<b>Première syntaxe :</b><br />';
echo "$avant => $après<br />";
echo "$x => ",strtr($x,$avant,$après),'<br />';
// deuxième syntaxe
$x = 'le bateau bleu et vert';
$correspondance = array('bleu'=>'rouge','vert'=>'jaune');
echo '<b>Deuxième syntaxe :</b><br />';
foreach($correspondance as $avant => $après)
  { echo "$avant => $après<br />"; }
echo "$x => ",strtr($x,$correspondance),'<br />';
?>
```

*Résultat*

```
Première syntaxe :
éèà => eea
cet été, à la plage => cet ete, a la plage
Deuxième syntaxe :
bleu => rouge
vert => jaune
le bateau bleu et vert => le bateau rouge et jaune
```

## Les expressions régulières - ereg[i] - ereg[i]_replace

PHP propose plusieurs fonctions qui permettent d'effectuer des recherches ou des remplacements dans une chaîne à l'aide d'un modèle appelé "expression régulière" décrivant l'élément recherché.

*Syntaxe*

```
entier ereg(chaîne rechercher, chaîne à_traiter[, tableau résultat])
entier eregi(chaîne rechercher, chaîne à_traiter[, tableau résultat])
chaîne ereg_replace(chaîne rechercher, chaîne remplacer, chaîne à_traiter)
chaîne eregi_replace(chaîne rechercher, chaîne remplacer, chaîne à_traiter)
```

`à_traiter`

Chaîne à traiter.

`rechercher`

Chaîne donnant le modèle (l'expression régulière) de l'élément recherché.

`remplacer`

Chaîne de remplacement.

`résultat`

Sous forme de tableau, liste des portions de la chaîne `à_traiter` qui correspondent au modèle recherché.

Les fonctions `ereg` et `eregi` recherchent, dans la chaîne `à_traiter`, s'il existe une chaîne qui correspond au modèle spécifié par `rechercher` (voir ci-après pour les règles). La fonction `ereg` est sensible à la casse (une majuscule est différente d'une minuscule) alors que `eregi` ne l'est pas. Ces fonctions retournent la longueur de l'occurrence trouvée si le modèle est trouvé et `FALSE` dans le cas contraire.

En complément, si le modèle le demande (voir ci-dessous pour les règles) et si un troisième paramètre est fourni, les différentes portions de la chaîne `à_traiter` qui correspondent au modèle seront stockées sous forme de tableau dans le troisième paramètre.

Les fonctions `ereg_replace` et `eregi_replace` effectuent la recherche sur le même principe et remplacent les différentes portions de la chaîne `à_traiter` qui correspondent au modèle, par la chaîne `remplacer`. La fonction `ereg_replace` est sensible à la casse alors que `eregi_replace` ne l'est pas. Si aucune occurrence n'est trouvée, la chaîne `à_traiter` est retournée inchangée.

Plusieurs caractères spéciaux peuvent être utilisés dans le modèle pour décrire l'élément recherché :

| Caractère spécial | Signification |
|---|---|
| | Si ^ est présent comme premier caractère du modèle, cela |

<table>
<tr><td>^</td><td>indique que la chaîne doit commencer par ce qui suit :<br>^abc : doit commencer par abc.</td></tr>
<tr><td>$</td><td>Si $ est présent comme dernier caractère du modèle, cela indique que la chaîne doit terminer par ce qui précède :<br>xyz$ : doit terminer par xyz.<br>^abcxyz$ : doit commencer par abcxyz et se terminer par abcxyz (bref être égal à abcxyz !).<br>Un modèle ne comportant ni ^ ni $ indique que le modèle est recherché n'importe où à l'intérieur de la chaîne :<br>abc :contient abc.</td></tr>
<tr><td>*</td><td>Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), peut être présente zéro, une ou plusieurs fois :<br>ab*c accepte ac, abc, abbc ...</td></tr>
<tr><td>+</td><td>Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), doit être présente une ou plusieurs fois :<br>ab+c accepte abc, abbc ..., mais refuse ac.</td></tr>
<tr><td>?</td><td>Indique que le caractère qui précède, ou la séquence qui précède (voir ci-après), peut être présente zéro ou une fois :<br>ab?c accepte ac et abc mais refuse abbc, abbbc ...</td></tr>
<tr><td>{x}<br>{x,}<br>{x,y}</td><td>Indique que le caractère qui précède, ou la séquence qui précède (voir ci- après), doit être présente exactement x fois ({x}) ou au minimum x fois ({x,}) ou entre x et y fois ({x,y}) :<br>ab{2}c n'accepte que abbc.<br>ab{2,4}c accepte abbc, abbbc, et abbbbc mais refuse abc (manque un b) ou abbbbbc (un b de trop).<br>ab{2,}c accepte abbc, abbbc, abbbbc, ... mais refuse abc (manque un b).</td></tr>
<tr><td>(...)</td><td>Permet de marquer une séquence recherchée, typiquement avec les symboles relatifs au nombre d'occurrences :<br>a(bc)*d accepte ad, abcd, abcbcd ... mais refuse abd ou acd (la séquence bc n'est pas trouvée).</td></tr>
<tr><td>x|y</td><td>Recherche de x ou de y (généralement utilisé avec les parenthèses pour éviter toute confusion) :<br>a(b|c)*d accepte ad, abd, acd, abcd, abbcd, accbd ... (en clair un nombre quelconque de b ou de c situés dans n'importe quel ordre entre un a et un d).</td></tr>
<tr><td>[...]</td><td>Permet de spécifier une série de caractères acceptés soit sous la forme c1c2...cn pour une liste exhaustive précise, soit sous la forme c1-c2 pour une plage de caractères, soit en mélangeant les deux :<br>[abcd] accepte un caractère parmi abcd (équivalent à (a|b|c|d)).<br>[a-z] accepte un caractère compris entre a et z.<br>[123a-zA-z] accepte un caractère compris entre a et z ou compris entre A et Z ou égal à 1 ou 2 ou 3.<br>[a-zA-z0-9] accepte un caractère compris entre a et z ou entre A et Z ou entre 0 et 9.</td></tr>
</table>

| | |
|---|---|
| | Une exclusion peut être spécifiée en mettant un ^ en premier caractère à l'intérieur des crochets :<br>[^0-9] refuse tout caractère compris entre 0-9.<br>[^abc] refuse les caractères a, b et c.<br>Compte tenu de sa signification particulière, le signe -, s'il est recherché en tant que tel, doit figurer en premier ou en dernier entre les crochets. |
| . | Indique un caractère quelconque :<br>a.b accepte toute séquence comportant un a suivi d'exactement un caractère quelconque suivi d'un b : axb, ayb, mais pas ab ni axyb.<br>a.{0,2}b accepte toute séquence comportant un a suivi de zéro à deux caractères quelconques suivi d'un b : ab, axb, , axyb, mais pas axyzb. |
| \ | Permet d'échapper les caractères spéciaux (^.[$()\|*+?{\) lorsque ceux-ci sont recherchés en tant que tels, sauf lorsqu'ils sont mentionnés entre crochets :<br>a*{2,4}b permet de rechercher les séquences commençant par un a suivi de 2 à 4 étoiles suivies d'un b (notez le *).<br>a[$*+]{2,4}b permet de rechercher les séquences commençant par un a suivi de 2 à 4 caractères pris parmi $, * et + suivis d'un b (pas besoin de \). |
| (...) | Les parenthèses ont une deuxième signification lorsqu'elles englobent tout le modèle ou des parties du modèle. Elles permettent de "capturer" dans la chaîne les séquences qui répondent au modèle ou à la portion de modèle entre parenthèses. Dans le cas de l'utilisation des fonctions ereg et eregi, les portions capturées sont stockées sous forme de tableau dans le troisième paramètre (s'il est spécifié) : la première ligne (indice 0) du tableau contient une copie de la chaîne correspondant à la totalité du modèle et les portions capturées sont stockées dans les lignes suivantes. Dans le cas de l'utilisation des fonctions ereg_replace et eregi_replace, les portions capturées peuvent être reprises dans la chaîne de remplacement grâce à la notation \\n, avec n entier (0 désignant la portion de la chaîne correspondant à la totalité du modèle et les numéros 1 à n les portions capturées). |
| [::] | Classe de caractères (voir ci-dessous). |

Les classes de caractères sont les suivantes :

[:alnum:]

Caractères alphanumériques

[:alpha:]

Caractères alphabétiques

[:blank:]

Caractères blancs

[:cntrl:]

Caractères de contrôle

[:digit:]

Chiffres

`[:graph:]`

Caractères graphiques

`[:lower:]`

Caractères alphabétiques minuscules

`[:print:]`

Caractères graphiques ou blancs

`[:punct:]`

Caractères de ponctuation

`[:space:]`

Espace, tabulation, nouvelle ligne, retour chariot

`[:upper:]`

Caractères alphabétiques majuscules

`[:xdigit:]`

Caractères des chiffres hexadécimaux

Les expressions régulières sont des outils très puissants pour effectuer des recherches dans une chaîne ou valider la conformité d'une saisie à certaines règles.

Les différents sites Web présentés dans le chapitre Introduction regorgent d'exemples de code utilisant les expressions régulières.

En complément des quelques exemples présentés ci-après, le chapitre Gérer les liens et les formulaires avec PHP illustrera l'utilisation des expressions régulières pour valider la saisie d'un utilisateur.

*Exemple 1*

```
<?php
// Vérifier qu'une chaîne commence par une lettre et
// est suivie d'au moins 3 lettres ou chiffres ou caractères
// spéciaux _(#*$).
$modèle = '^[a-z][a-z0-9_#*$]{3,}';
// Tableau contenant les chaînes à tester.
$chaînes[] = 'A0_#b*1$2'; // OK;
$chaînes[] = '0_#b*1$2';  // ne commence pas par une lettre;
$chaînes[] = 'A0_';       // longueur insuffisante;
$chaînes[] = 'A0_€#';     // caractère invalide;
// Utilisation de eregi (insensible à la casse).
foreach ($chaînes as $chaîne) {
  if (eregi($modèle,$chaîne) === FALSE) { // test : ===
    echo "$chaîne => pas OK<br />";
  } else {
    echo "$chaîne => OK<br />";
  }
}?>
```

*Résultat*

```
A0_#b*1$2 => OK
0_#b*1$2 => pas OK
A0_ => pas OK
A0_€# => pas OK
```

Quelques explications sur l'expression régulière utilisée (`^[a-z][a-z0-9_#*$]{3,}`) :

- `eregi` est utilisée afin de ne pas faire de différence entre majuscules et minuscules.
- `^` = commence par ...
- `[a-z]` = une lettre entre `a` et `z` (ou `A` et `Z` car `eregi` est utilisée) ...

- [a-z0-9_#*$]{3,} = suivi d'au moins 3 ({3,}) caractères parmi ceux indiqués : a à z (et donc A à Z), 0 à 9 et les caractères _#*$

_Exemple_

```
<?php
// Vérifier qu'une chaîne à une structure conforme à celle
// d'une date au format [J]J/[M]M/AAAA et récupérer les
// 3 composantes jour, mois et année.
$modèle = '^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$';
// Tableau contenant les chaînes à tester.
$dates[] = '21/09/2001'; // OK
$dates[] = '1/2/2001';   // OK
$dates[]  = '21/09/01';   // année incomplète
// Utilisation de ereg.
foreach ($dates as $date) {
  $ok = ereg($modèle,$date,$résultat);
  if ($ok) {
    echo "$date valide.<br />";
    echo "- jour = $résultat[1]<br />";
    echo "- mois = $résultat[2]<br />";
    echo "- année = $résultat[3]<br />";
  } else {
    echo "$date invalide.<br />";
  }
}
?>
```

_Résultat_

```
21/09/2001 valide.
- jour = 21
- mois = 09
- année = 2001
1/2/2001 valide.
- jour = 1
- mois = 2
- année = 2001
21/09/01 invalide.
```

Quelques explications sur l'expression régulière utilisée (^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$) :

- ^ = commence par ...
- [0-9]{1,2} = un ou deux chiffres ...
- / = suivi du caractère "/" ...
- [0-9]{1,2} = suivi de un ou deux chiffres ...
- / = suivi du caractère « / » ...
- [0-9]{4} = suivi de 4 chiffres ...
- $ = suivi de ... rien du tout ! La chaîne doit se terminer immédiatement.

---

Ce test permet de vérifier qu'une chaîne censée contenir une date est bien formée. Il convient ensuite de vérifier, avec la fonction checkdate par exemple, que les trois composantes correspondent bien à une date valide.

---

_Exemple 3_

```
<?php
// Utiliser ereg_replace pour réorganiser une chaîne.
// En l'occurrence, il s'agit de transformer une date
// au format JJ/MM/AAAA en date au format AAAA-MM-JJ
```

```
$avant = ’26/08/1966’;
$après = ereg_replace(
             ’^([0-9]{2})/([0-9]{2})/([0-9]{4})$’,
             ’\\3-\\2-\\1’,
             $avant);
echo "$avant => $après";
?>
```

*Résultat*

```
26/08/1966 => 1966-08-26
```

Dans la chaîne de remplacement, les séquences `\\n` désignent les trois portions capturées par les parenthèses :

- `\\1` = premier (`[0-9]{2}`)= 26
- `\\2` = deuxième (`[0-9]{2}`)= 08
- `\\3` = (`[0-9]{4}`)= 1966

Le résultat de la recherche, qui est égal à la chaîne complète dans ce cas, est donc ensuite remplacé par la chaîne ’`\\3-\\2-\\1`’ soit `1966-08-26`.

# Manipuler les dates

PHP ne gère pas les dates avec un type de donnée spécifique. Néanmoins, des dates peuvent être manipulées, soit sous la forme d'une chaîne de caractères, soit sous la forme d'un *timestamp* Unix (correspondant au nombre de secondes écoulées depuis le 1er janvier 1970 01:00:00).

Plusieurs fonctions permettent de manipuler les dates sous l'une ou l'autre de ces formes :

| **Nom** | **Rôle** |
|---|---|
| `checkdate` | Vérifie que trois entiers représentant le jour, le mois et l'année correspondent à une date valide. |
| `date` | Convertit en chaîne une date donnée sous la forme d'un *timestamp* Unix. |
| `strftime` | Convertit en chaîne une date donnée sous la forme d'un *timestamp* Unix, en utilisant des caractéristiques locales. |
| `getdate` | Stocke dans un tableau les différentes composantes d'une date donnée sous la forme d'un *timestamp* Unix. |
| `time` | Donne le *timestamp* Unix actuel. |
| `mktime` | Crée un *timestamp* Unix à partir des différentes composantes d'une date. |
| `microtime` | Donne le *timestamp* Unix actuel accompagné du nombre de microsecondes écoulées depuis la dernière seconde. |
| `idate` | Donne les composantes d'une date fournie sous la forme d'un *timestamp* Unix. |

Depuis la version 5.1.0, il existe des fonctions qui manipulent des dates sous la forme d'un objet (voir notamment le fonction `date_create` dans la documentation).

## checkdate

La fonction `checkdate` vérifie que trois entiers représentant le jour, le mois et l'année correspondent à une date valide.

*Syntaxe*

```
booléen checkdate(entier mois, entier jour, entier année)
```

`mois`

Numéro du mois (1 à 12).

`jour`

Numéro du jour (1 à 31).

`année`

Année (1 à 32767).

La fonction `checkdate` retourne `TRUE` si la date construite avec les trois composantes est valide et `FALSE` dans le cas contraire. Cette fonction tient compte des années bissextiles.

*Exemple*

```
<?php
$jour = 26; $mois = 8 ; $année = 1966;
echo "$jour/$mois/$année => ",
      var_dump(checkdate($mois,$jour,$année)),'<br />';
$jour = 29; $mois = 2 ; $année = 2000;
echo "$jour/$mois/$année => ",
      var_dump(checkdate($mois,$jour,$année)),'<br />';
$jour = 29; $mois = 2 ; $année = 2001;
echo "$jour/$mois/$année => ",
      var_dump(checkdate($mois,$jour,$année)),'<br />';
?>
```

*Résultat*

```
26/8/1966 => bool(true)
29/2/2000 => bool(true)
29/2/2001 => bool(false)
```

### date

La fonction `date` convertit en chaîne une date donnée sous la forme d'un *timestamp* Unix.

*Syntaxe*

```
chaîne date(chaîne format[, entier timestamp])
```

`format`

Format de conversion.

`timestamp`

Timestamp à convertir (par défaut le timestamp actuel).

Le format peut être spécifié à l'aide des caractères suivants (non exhaustif) :

| Caractère | Signification |
|---|---|
| d | Numéro du jour du mois, sur deux chiffres (01 à 31) |
| j | Numéro du jour du mois, sur un ou deux chiffres (1 à 31) |
| m | Numéro du mois, sur deux chiffres (01 à 12) |
| n | Numéro du mois, sur un ou deux chiffres (1 à 12) |
| Y | Année sur quatre chiffres (2001 par exemple) |
| y | Année sur deux chiffres (01 par exemple) |
| H | Heure, au format 24h, sur deux chiffres (00 à 23) |
| i | Minutes sur deux chiffres (00 à 59) |
| s | Secondes sur deux chiffres (00 à 59) |
| z | Numéro du jour de l'année (1 à 365) |
| w | Numéro du jour de la semaine (0 = dimanche à 6 = samedi) |
| N | Numéro du jour de la semaine selon la norme ISO 8601 (1 = lundi à 7 = dimanche). Ajouté dans la version |

| | |
|---|---|
| | 5.1.0. |
| W | Numéro de la semaine dans l'année selon la norme ISO 9601 |
| D | Trois premières lettres du nom du jour de la semaine, en anglais |
| r | Format de date de la RFC 822 (par exemple, "Thu, 20 Sep 2001 15:47:00 +0200") |

En cas de besoin, ces différents caractères peuvent être échappés avec un anti-slash (\).

Depuis la version 5.1.1, des formats standards peuvent être spécifiés à l'aide de constantes, par exemple :

`DATE_ISO8601`

ISO-8601 (exemple : 2007-07-13T17:53:10+0200)

`DATE_RFC822`

RFC 822 (exemple : Fri, 13 Jul 07 17:53:10 +0200)

`DATE_RSS`

RSS (exemple : Fri, 13 Jul 2007 17:53:10 +0200)

*Exemple*

```
<?php
// sans deuxième paramètre = utilisation du timestamp courant
echo 'Date au format JJ/MM/AAAA : ', date('d/m/Y'),'<br />';
echo 'Heure : ', date('H:i:s'),'<br />';
echo 'Unix a fêté sa milliardième seconde le ',
      date('d/m/Y à H:i:s',1000000000),'<br />';
?>
```

*Résultat*

```
Date au format JJ/MM/AAAA : 28/01/2008
Heure : 20:58:12
Unix a fêté sa milliardième seconde le 09/09/2001 à 03:46:40
```

## strftime

La fonction `strftime` convertit en chaîne une date donnée sous la forme d'un timestamp Unix, en utilisant des caractéristiques locales.

*Syntaxe*

```
chaîne strftime(chaîne format[, entier timestamp])
```

`format`

Format de conversion.

`timestamp`

*Timestamp* à convertir (par défaut le timestamp actuel).

À la différence de la fonction `date`, `strftime` utilise les caractéristiques linguistiques locales (celles du serveur).

Le format peut être spécifié à l'aide des caractères suivants (non exhaustif) :

| Caractère | Signification |
|---|---|
| %d | Jour du mois, sur deux chiffres (01 à 31) |

| | |
|---|---|
| %m | Numéro du mois, sur deux chiffres (01 à 12) |
| %y | Année sur deux chiffres (01 par exemple) |
| %Y | Année sur quatre chiffres (2001 par exemple) |
| %H | Heure, au format 24h |
| %M | Minutes sur deux chiffres (00 à 59) |
| %S | Secondes sur deux chiffres (00 à 59) |
| %j | Numéro du jour de l'année sur trois chiffres (001 à 365) |
| %w | Numéro du jour de la semaine (0 = dimanche à 6 = samedi) |
| %u | Numéro du jour de la semaine (1 = lundi à 7 = dimanche) |
| %a | Nom abrégé du jour de la semaine |
| %A | Nom complet du jour de la semaine |
| %b ou %h | Nom abrégé du mois |
| %B | Nom complet du mois |
| %U | Numéro de semaine dans l'année (en considérant le premier dimanche de l'année comme le premier jour de la première semaine) |
| %W | Numéro de semaine dans l'année (en considérant le premier lundi de l'année comme le premier jour de la première semaine) |
| %V | Numéro de la semaine dans l'année selon la norme ISO 9601 |
| %Z | Fuseau horaire, ou nom ou abréviation |
| %c | Format par défaut pour la date et l'heure |
| %x | Format par défaut pour la date seule |
| %X | Format par défaut pour l'heure seule |
| %R | Identique à %H:%M |
| %T | Identique à %H:%M:%S |
| %t | Tabulation |
| %n | Retour à la ligne |
| %% | un caractère '%' littéral |

Tous les symboles ne sont pas forcément supportés sur toutes les plates-formes.

Les caractéristiques linguistiques locales peuvent être lues et modifiées par l'intermédiaire de la fonction `setlocale`.

*Syntaxe*

```
chaîne setlocal(mixte catégorie, chaîne langue)
```

| | |
|---|---|
| `catégorie` | Fonctionnalité concernée par les caractéristiques linguistiques locales définies à l'aide d'une des constantes suivantes :<br>`LC_COLLATE` : comparaison de chaîne avec la fonction `strcoll` ;<br>`LC_CTYPE` : classification et conversions (fonction `stroupper` par exemple) ;<br>`LC_NUMERIC` : séparateur décimal ;<br>`LC_MONETARY` : symbole monétaire (fonction `localeconv`) ;<br>`LC_TIME` : formats de date (fonction `strftime`) ;<br>`LC_ALL` : toutes les précédentes. |
| `langue` | Code de la langue devant gouverner la fonctionnalité associée. Extrait des valeurs possibles :<br>`du` : Hollande ;<br>`fr` : France ;<br>`ge` : Allemagne ;<br>`it` : Italie ;<br>`ru` : Russie ;<br>`sp` : Espagne ;<br>`sw` : Suède ;<br>`en` : Royaume-Uni ;<br>`us` : Etats-Unis. |

La fonction `setlocal` retourne la nouvelle valeur ou `FALSE` en cas d'erreur. Si le deuxième paramètre est égal à `"0"`, `setlocal` retourne simplement la valeur courante. S'il est égal à `NULL` ou une chaîne vide, `setlocale` prend la valeur correspondant à l'environnement du système d'exploitation.

*Exemple*

```
<?php
echo 'Date/Heure : ',
      strftime('%d/%m/%Y - %H:%M:%S '),'<br />';
setlocale(LC_ALL,'fr');
echo 'Format long (français) : ',
      strftime('%A %d %B %Y'),'<br />';
setlocale(LC_ALL,'en');
echo 'Format long (anglais) : ',
      strftime('%A %d %B %Y'),'<br />';
?>
```

*Résultat*

```
Date/Heure: 28/01/2008 - 21:21:48
Format long (français) : lundi 28 janvier 2008
Format long (anglais) : Monday 28 January 2008
```

## getdate

La fonction `getdate` stocke, dans un tableau, les différentes composantes d'une date donnée sous la forme d'un

*timestamp* Unix.

*Syntaxe*

```
tableau getdate(entier timestamp)
```

```
timestamp
```

*Timestamp* à utiliser (par défaut le timestamp actuel).

La fonction `getdate` retourne un tableau associatif comportant les clés suivantes :

| Clé | Valeur |
|---|---|
| `seconds` | Secondes (0 à 59) |
| `minutes` | Minutes (0 à 59) |
| `hours` | Heures (0 à 24) |
| `mday` | Numéro du jour du mois |
| `wday` | Numéro du jour de la semaine (de 0 = dimanche à 6 = samedi) |
| `mon` | Numéro du mois (1 à 12) |
| `year` | Année |
| `yday` | Numéro du jour dans l'année (1 à 365) |
| `weekday` | Nom du jour de la semaine |
| `month` | Nom du mois |
| `0` | Le timestamp |

*Exemple*

```
<?php
$date = getdate(); // maintenant
foreach($date as $clé => $valeur) {
  echo "$clé => $valeur<br />";
}
?>
```

*Résultat*

```
seconds => 54
minutes => 28
hours => 21
mday => 28
wday => 1
mon => 1
year => 2008
yday => 27
weekday => Monday
month => January
0 => 1201552134
```

## time

La fonction `time` donne le *timestamp* Unix actuel.

*Syntaxe*

```
entier time()
```

*Exemple*

```
<?php
$ts = time();
echo "timestamp Unix actuel = $ts";
?>
```

*Résultat*

```
timestamp Unix actuel = 1201552164
```

**mktime**

La fonction `mktime` crée un *timestamp* Unix à partir des différentes composantes d'une date.

*Syntaxe*

```
entier mktime([entier heure[,entier minutes[,entier secondes[,entier mois[,
entier jour[,entier année]]]]]])
```

`heure`

Heure (0 à 23)

`minutes`

Minutes (0 à 59)

`secondes`

Secondes (0 à 59)

`mois`

Mois (1 à 12)

`jour`

Jour (1 à 31)

`année`

Année sur 2 ou 4 chiffres. L'année, sur 4 chiffres, doit être comprise entre 1902 (1970 avant la version 5.1.0) et 2038. Nous déconseillons de spécifier l'année sur 2 chiffres (différence de comportement entre les versions).

Les paramètres omis prennent leur valeur actuelle.

La fonction `mktime` a la particularité intéressante de corriger les valeurs incorrectes en effectuant un calcul de date intelligent. Exemples :

- le 35/12/2001 sera corrigé en 31/12/2001 + 4 jours = 04/01/2002
- le 30/14/2001 sera corrigé en 30/12/2001 + 2 mois = 30/02/2002 qui lui-même est corrigé en 28/02/2002 + 2 jours = 02/03/2002

Ce fonctionnement est très pratique pour réaliser des calculs sur les dates.

*Exemple*

```
<?php
$ts = mktime();
echo 'mktime() = maintenant = ',
      date('d/m/Y - H:i:s',$ts),'<br />';
```

```
$ts = mktime(0,0,0,8,26,1966);
echo 'mktime(0,0,0,8,26,1966) = ',
      date('d/m/Y - H:i:s',$ts),'<br />';
$ts = mktime(0,0,0,8,26+20000,1966);
echo '20000 jours après le 26/08/1966 = ',
      date("d/m/Y",$ts),"<BR>";
?>
```

*Résultat*

```
mktime() = maintenant = 28/01/2008 - 21:39:51
mktime(0,0,0,8,26,1966) = 26/08/1966 - 00:00:00
20000 jours après le 26/08/1966 = 29/05/2021
```

## microtime

La fonction `microtime` retourne le *timestamp* Unix actuel avec la fraction de seconde en microsecondes.

*Syntaxe*

```
mixte microtime([booléen type_réel])
```

`type_réel`

Booléen indiquant si la fonction doit retourner un nombre réel.

Sans paramètre (ou si le paramètre est évalué à `FALSE`), la fonction retourne une chaîne donnant les microsecondes suivies d'un espace et du *timestamp* Unix actuel. Si le paramètre est évalué à `TRUE`, la fonction retourne un nombre réel.

*Exemple*

```
<?php
// Affichage de microtime sous la forme d'une chaîne.
echo microtime().'<br />';
// Affichage de microtime sous la forme d'un réel.
echo microtime(TRUE).'<br />';
// Pour ne conserver que les micosecondes, le plus
// simple est de transformer la chaîne en réel.
echo (float) microtime().'<br />';
?>
```

*Résultat*

```
0.45474800 1201552264
1201552264.45
0.454881
```

## idate

La fonction `idate` retourne les différentes composantes (année, mois, etc.) d'un timestamp Unix.

*Syntaxe*

```
entier idate(caractère composante [,entier timestamp])
```

`composante`

Caractère indiquant la composante souhaitée (voir ci-après).

`timestamp`

Timestamp à utiliser (par défaut le timestamp actuel).

La fonction retourne un entier correspondant à la composante demandée.

La composante peut être spécifiée à l'aide d'un des caractères suivants (liste non exhaustive) :

| **Caractère** | **Signification** |
|---|---|

| | |
|---|---|
| Y | Année sur 4 chiffres |
| z | Jour de l’année |
| y | Année sur 2 chiffres |
| m | Numéro du mois |
| D | Numéro du jour du mois |
| H | Heure sur 24 |
| i | Minutes |
| s | Secondes |
| t | Nombre de jours dans le mois |
| W | Numéro de semaine de l’année |
| w | Jour de la semaine (0 pour dimanche) |

*Exemple*

```
<?php
// Affichage de la date/heure courante pour contrôle
echo strftime('%d/%m/%Y - %H:%M:%S'),'<br />';
// Extraction de différentes composantes
$composantes = str_split('YmdHistwW');
foreach($composantes as $composante) {
  echo "$composante = ",idate($composante),'<BR>';
}
?>
```

*Résultat*

```
28/01/2008 - 21:33:12
Y = 2008
m = 1
d = 28
H = 21
i = 33
s = 12
t = 31
w = 1
W = 5
```

# Générer un nombre aléatoire

La fonction rand permet de générer des nombres aléatoires.

*Syntaxe*

```
entier rand([entier min[, entier max]])
```

min et max

Bornes des nombres aléatoires à générer. Valeur par défaut de min = 0. Valeur par défaut de max = une valeur donnée par la fonction getrandmax.

La fonction rand retourne un nombre aléatoire entier compris entre la borne minimum et la borne maximum incluses.

```
<?php
// Génération de nombres aléatoires.
echo rand().'<br />';
echo rand().'<br />';
echo rand(1,100).'<br />';
echo rand(1,100).'<br />';
?>
```

*Résultat*

```
214120785
257944955
68
65
```

# Créer un identifiant unique

Dans certaines situations, il peut être nécessaire de générer des identifiants uniques.

PHP propose la fonction `uniqid` pour générer des identifiants uniques.

*Syntaxe*

```
chaîne uniqid()([chaîne préfixe [, booléen plus_unique]])
```

`préfixe`

Préfixe à ajouter à l'identifiant.

Mettre une chaîne vide ou ne rien mettre si vous ne souhaitez pas de préfixe.

`plus_unique`

Si ce paramètre est positionné à `TRUE`, des données supplémentaires sont ajoutées à la fin de la valeur retournée pour obtenir un identifiant plus long et plus difficilement identifiable.

La fonction `uniqid` retourne une chaîne de treize caractères, ou vingt-trois si le paramètre `plus_unique` est à `TRUE` (sans compter le préfixe), calculée à partir de l'heure courante en microsecondes.

*Exemple*

```
<?php
echo uniqid(),'<br />';
echo uniqid(),'<br />';
echo uniqid('abc'),'<br />';
echo uniqid('',TRUE) ,'<br />';
?>
```

*Résultat*

```
47864520950dd
47864520958b1
abc47864520960b6
47864520960bf5.27165445
```

Cet exemple montre que l'identifiant généré est bien unique, même si la différence entre deux appels successifs est faible. Du coup, l'identifiant généré peut être jugé insuffisamment aléatoire et un peu trop déterministe.

Une technique classique consiste alors à hacher l'identifiant généré. La fonction `md5` permet de le faire très facilement, en utilisant une méthode MD5.

*Syntaxe*

```
chaîne md5(chaîne valeur)
```

*valeur*

Chaîne à hacher.

La fonction `md5` retourne la chaîne hachée.

*Exemple*

```
<?php
echo md5('olivier'),'<br />';
echo md5(uniqid()),'<br />';
echo md5(uniqid()),'<br />';
?>
```

*Résultat*

```
d3ca5dde60f88db606021eeba2499c02
694fb65aee42c22b371bcd9f9b3cfc3b
```

```
b69e0eba91d77fe9b6b6a4f80e3d4acd
```

La combinaison des fonctions `uniqid` et `md5` donne un identifiant qui comporte 32 caractères. Cet identifiant est maintenant plus aléatoire et moins facile à déterminer.

Pour les paranoïaques de la sécurité, il est possible d’aller encore plus loin en utilisant en plus un préfixe aléatoire.

*Exemple*

```
<?php
echo md5(uniqid(rand())),'<br />';
echo md5(uniqid(rand())),'<br />';
?>
```

*Résultat*

```
0615709d67387b9c91df4d8f9eb5a92a
b6c5ec69c0d4d0ed76c5f48e95857ac6
```

# Gérer les "guillemets magiques" ("magic quotes")

## 1. Principe

PHP propose une fonctionnalité, appelée "magic quotes" ("guillemets magiques") dont l'objectif principal est de résoudre un problème lié à l'enregistrement des données dans une base de données en effectuant un encodage sur les données venant de "l'extérieur" :

- données saisies dans un formulaire, transmises dans une URL ou envoyées par un cookie ;
- données lues dans une base de données ou un fichier.

Cette "fonctionnalité" peut rapidement devenir un casse-tête si elle est mal prise en compte.

Lorsque la fonctionnalité "magic quotes" est active, les caractères apostrophe ('), guillemet ("), anti-slash (\) et nul (`\0`) sont automatiquement protégés par un anti-slash (\).

*Exemple*

| Donnée initiale | Donnée récupérée dans le script |
|---|---|
| `c'est l'été` | `c\'est l\'été` |

Deux directives permettent d'activer (valeur `on`) ou de désactiver (valeur `off`) les "guillemets magiques".

| Directive | Données concernées |
|---|---|
| `magic_quotes_gpc` | Données saisies dans un formulaire, transmises dans une URL ou envoyées par un cookie. |
| `magic_quotes_runtime` | Données lues dans une base de données MySQL ou dans un fichier texte. |

Le suffixe `gpc` de la directive correspond à Get, Post et Cookie.

Ce principe d'encodage "magic quotes" est intéressant, notamment devant l'apostrophe, si les données sont destinées à être enregistrées dans une base SQL. En effet, en SQL, le délimiteur de chaîne de caractères est l'apostrophe (cf. chapitre Introduction à MySQL - Apprendre les bases du langage SQL) ; si une requête envoie la chaîne `'c'est l'été'` à la base de données, cette dernière va interpréter `'c'` comme une chaîne et ne saura pas quoi faire du reste (`est l'été'`) : un message d'erreur sera retourné par la base.

Pour régler ce problème, il faut indiquer à la base que les apostrophes à l'intérieur de la chaîne ne sont pas des délimiteurs de la chaîne, généralement en les faisant précéder d'un caractère "magique" ("d'échappement") : il s'agit de l'anti-slash (\) pour MySQL. La bonne valeur à envoyer est donc `'c\'est l\'été'` pour MySQL.

Si la directive `magic_quotes_sybase` est à `on`, le caractère d'échappement est remplacé par l'apostrophe ; ce caractère d'échappement est utilisé par d'autres bases de données comme Oracle ou Microsoft SQL Server.

La fonctionnalité "magic quotes" est intéressante si la donnée saisie est destinée à être enregistrée dans une base MySQL (l'encodage est automatique), mais pose des problèmes si la donnée saisie est destinée à être simplement affichée dans une page HTML (l'encodage est inutile).

Dans la pratique, cette fonctionnalité "magic quotes" pose deux problèmes :

- Si la donnée est à la fois destinée à être affichée dans la page et enregistrée dans une base de données, il y a un problème : avec l'affichage si "magic quotes" est actif et avec l'enregistrement dans la base de données si "magic quotes" est inactif.
- Le développeur ne maîtrise pas forcément la configuration de PHP dans l'environnement d'exploitation : il peut donc écrire du code qui fonctionne dans son environnement de développement (avec une certaine configuration de "magic quotes"), mais qui risque de ne plus fonctionner dans l'environnement d'exploitation où la configuration est peut être différente.

L'équipe de développement de PHP recommande de désactiver la fonctionnalité "magic quotes" : la directive `magic_quotes_gpc` est à `off` dans le fichier de configuration recommandé (`php.ini-recommended`). Dans une future version (sans doute la version 6), la fonctionnalité "magic quotes" sera définitivement désactivée.

Pour être conforme à cette recommandation et préparer le futur, nous vous conseillons de désactiver la fonctionnalité "magic

quotes", dans la mesure du possible.

Néanmoins, dans ce livre, nous verrons comment gérer proprement cette fonctionnalité et écrire un code indépendant de la configuration.

## 2. Fonctions relatives aux "guillemets magiques"

PHP propose plusieurs fonctions qui permettent de gérer correctement la fonctionnalité "magic quotes" :

`get_magic_quotes_gpc`

Permet de connaître la valeur de la directive `magic_quotes_gpc`.

`get_magic_quotes_runtime`

Permet de connaître la valeur de la directive `magic_quotes_runtime`.

`set_magic_quotes_runtime`

Modifie la valeur de la directive `magic_quotes_runtime` en cours de script.

`addslashes`

Ajoute une protection "magic quotes" dans une chaîne de caractères.

`stripslashes`

Supprimer la protection "magic quotes" d'une chaîne de caractères.

---

La directive `magic_quotes_gpc` ne peut pas être modifiée en cours de script.

---

### get_magic_quotes_gpc

La fonction `get_magic_quotes_gpc` permet de connaître la valeur de la directive `magic_quotes_gpc`.

*Syntaxe*

```
entier get_magic_quotes_gpc()
```

La fonction `get_magic_quotes_gpc` retourne `0` si la directive `magic_quotes_gpc` est à `off`, ou `1` sinon.

### get_magic_quotes_runtime et set_magic_quotes_runtime

La fonction `get_magic_quotes_runtime` permet de connaître la valeur de la directive `magic_quotes_runtime` et la fonction `set_magic_quotes_runtime` de la modifier en cours de script.

*Syntaxe*

```
entier get_magic_quotes_runtime()
booléen set_magic_quotes_runtime(entier valeur)
```

`valeur`

Nouvelle valeur de la directive `magic_quotes_runtime` : `0` = désactivé (`off`), `1` = activé (`on`).

La fonction `get_magic_quotes_runtime` retourne `0` si l'option est désactivée et `1` si elle est activée.

La fonction `set_magic_quotes_runtime` retourne `TRUE` si le changement s'est effectué et `FALSE` sinon.

### addslashes et stripslashes

Les fonctions `addslashes` et `stripslashes` permettent d'ajouter ou de supprimer la protection "magic quotes" dans une chaîne de caractères.

*Syntaxe*

```
chaîne addslashes(chaîne texte)
chaîne stripslashes(chaîne texte)
```

`texte`

Chaîne concernée.

La fonction addslashes prend en paramètre une chaîne de caractères et retourne cette chaîne en ajoutant un anti-slash (\) devant les caractères apostrophe ('), guillemet ("), anti slash (\) et nul (\0).

La fonction stripslashes est l'inverse de la fonction addslashes : elle prend en paramètre une chaîne de caractères et retourne cette chaîne en supprimant l'anti-slash (\) devant les caractères apostrophe ('), guillemet ("), anti slash (\) et nul (\0).

Dans les deux cas, le caractère d'échappement utilisé est l'apostrophe (') si la directive magic_quotes_sybase est à on.

*Exemple*

```
<?php
$texte = "c'est l'été";
$texte = addslashes($texte);
echo $texte,'<br />';
$texte = stripslashes($texte);
echo $texte,'<br />';
?>
```

*Résultat (en supposant que magic_quotes_sybase = off)*

```
c\'est l\'été
c'est l'été
```

Ces fonctions vont être utiles dans les deux cas suivants :

- Si vous avez une donnée dans une variable, qui n'a pas déjà été encodée, et que cette donnée doit être enregistrée dans la base, il est conseillé d'appeler addslashes pour effectuer l'encodage adapté.
- Si vous avez une donnée dans une variable, qui a été encodée (manuellement ou automatiquement), et que vous souhaitez l'afficher dans une page HTML, il faut appeler stripslashes pour enlever l'encodage.

Il ne faut pas appeler addslashes ou stripslashes au hasard sur des variables déjà encodées en ce qui concerne addslashes ou non encodées en ce qui concerne stripslashes.

*Exemple*

```
<?php
// Utilisation de la fonction addslashes sur une donnée déjà
// encodée.
echo addslashes("c\'est l\'été"),'<br />';
// Utilisation de la fonction stripslashes sur une donnée non
// encodée :
// - Cas 1 : pas grave
echo stripslashes("c'est l'été"),'<br />';
// - Cas 2 : mauvais
echo stripslashes('d:\photos\identité.jpg'),'<br />';
?>
```

*Résultat (en supposant que magic_quotes_sybase = off)*

```
c\\\'est l\\\'été
c'est l'été
d:photosidentité.jpg
```

Il est donc important de ne pas appeler addslahes systématiquement avant un enregistrement dans la base, ou stripslashes systématiquement avant un affichage dans une page HTML. Il faut savoir d'où vient la donnée et si elle est ou non déjà encodée par la fonctionnalité "magic quotes". Cette information peut être obtenue grâce à la fonction get_magic_quotes_gpc présentée précédemment.

## 3. Gestion intelligente et portable

Il existe deux situations simples si vous maîtrisez la configuration de PHP :

- Vous êtes certain que magic_quotes_gpc = off : dans ce cas, vous pouvez afficher sans souci les données en provenance d'un formulaire, d'une URL ou d'un cookie (pas besoin d'appeler stripslashes) et vous devez appeler addslashes avant tout enregistrement dans la base.

- Vous êtes <u>certain</u> que `magic_quotes_gpc = on` et que `magic_quotes_sybase` est adapté à la base que vous utilisez (`off` pour MySQL par exemple) : dans ce cas, vous devez appeler `stripslashes` avant tout affichage des données en provenance d'un formulaire, d'un URL ou d'un cookie, mais vous pouvez les enregistrer directement dans la base. Par contre un problème peut se poser pour des données ayant une autre source. Si vous initialisez une variable du type `$valeur = "c'est l'été"`, cette variable va contenir une information qui n'a pas subi l'encodage automatique de la fonctionnalité "magic quotes" ; à un moment ou un autre, il faudra encoder la donnée avec `addslashes` si elle doit être enregistrée dans la base ...

Si vous ne maîtrisez pas la configuration de PHP et que vous souhaitez avoir un code portable, il faut procéder autrement.

Voici la solution que nous préconisons :

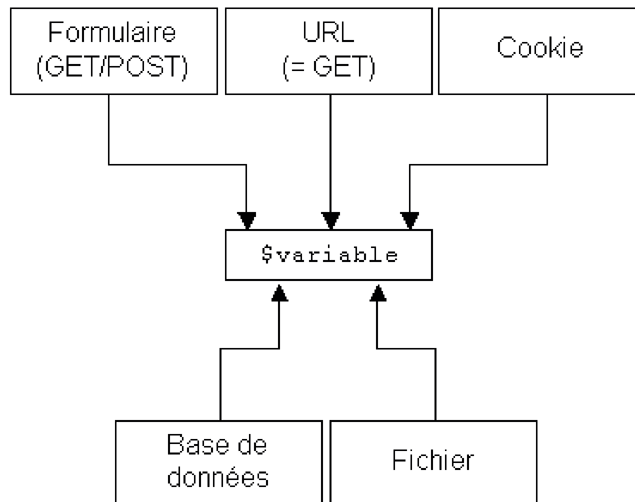
- Décidez une fois pour toute que toute variable PHP contient une donnée qui n'est pas protégée (approche cohérente avec les évolutions à venir de PHP).

- Faites le traitement nécessaire, lors de l'affectation d'une variable par une donnée qui vient de l'extérieur pour s'assurer qu'elle ne contient pas d'encodage "magic quotes".

- Ne faites rien vis-à-vis de l'encodage "magic quote" lors de l'affichage d'une variable dans une page HTML (les variables ne contiennent pas de caractères d'échappement superflus).

- Faites l'encodage nécessaire lors de l'enregistrement d'une variable dans une base.

Le schéma suivant présente les sources de données extérieures qu'il va falloir gérer au moment de l'alimentation des variables :




## Donnée en provenance d'un formulaire, d'une URL ou d'un cookie

Comme nous l'avons dit, tout ce qui vient d'un formulaire (méthode `GET` ou `POST`), d'une URL (méthode `GET`) ou d'un cookie, et que nous appellerons désormais données GPC, est sous l'influence de la directive `magic_quotes_gpc` : au moment de l'alimentation d'une variable à partir d'une de ces sources, pour respecter notre principe de fonctionnement, il faut enlever l'encodage "magic quotes" si "magic quotes" est actif et ne rien faire sinon.

Un tel traitement est possible grâce aux fonctions `get_magic_quotes_gpc` et `stripslashes`. Il suffit d'écrire une fonction qui sera systématiquement appelée lors de la récupération d'une donnée GPC pour enlever l'encodage "magic quotes" si ce dernier est actif.

*<u>Exemple</u>*

```
<?php
function  supprimer_encodage_MQ($valeur) {
  // Si magic quotes est actif, retourner
  // la valeur après suppression de l'encodage
  // (=> appel à stripslashes), sinon, retourner
  // la valeur.
  return
    (get_magic_quotes_gpc())?stripslashes($valeur):$valeur;
}
?>
```

---

Appelez cette fonction uniquement sur des données GPC pour ne pas risquer d'enlever des caractères d'échappement qui ne sont pas liés à un encodage "magic quotes".

---

Dans le chapitre Gérer les liens et les formulaires avec PHP, nous reviendrons sur ce sujet et nous présenterons une méthode alternative permettant de supprimer en un seul appel la protection éventuelle de toutes les données GPC.

### Donnée en provenance de MySQL ou d'un fichier texte

Pour les données en provenance de MySQL ou d'un fichier texte, le traitement est beaucoup plus simple puisqu'il est possible de modifier en cours de script la valeur de la directive `magic_quotes_runtime`.

Un simple appel à `set_magic_quotes_runtime(0)` permet de garantir qu'il n'y aura pas de protection "magic quotes" dans les données lues par la suite dans une base MySQL (cf. chapitre Accéder à une base de données MySQL) ou dans un fichier texte (cf. dans ce chapitre Accéder à une base de données MySQL - Manipuler les fichiers sur le serveur).

### Enregistrement dans une base de données MySQL

Pour l'enregistrement dans la base de données, nous verrons dans le chapitre 11 comment traiter de manière générique les données avant de les envoyer à la base pour éviter les problèmes.

# Manipuler les fichiers sur le serveur

## 1. Fonctions utiles

PHP propose un grand nombre de fonctions permettant de manipuler les fichiers sur le serveur.

Les fonctions les plus utiles sont les suivantes :

| Nom | Rôle |
|---|---|
| `fopen` | Ouvrir un fichier |
| `fclose` | Fermer un fichier |
| `fread` | Lire le contenu d'un fichier (dans une chaîne) |
| `file` | Lire le contenu d'un fichier (dans un tableau) |
| `fwrite` | Écrire dans un fichier |
| `file_get_contents` | Ouvrir, lire et fermer un fichier |
| `file_put_contents` | Ouvrir, écrire et fermer un fichier |
| `readfile` | Affiche le contenu d'un fichier directement sur la sortie |
| `copy` | Copier un fichier |
| `unlink` | Supprimer un fichier |
| `rename` | Renommer un fichier |
| `file_exists` | Tester l'existence d'un fichier |
| `filesize` | Lire la taille d'un fichier |

Certaines de ces fonctions vont prendre comme paramètre un nom de fichier ou de répertoire. Sur la plate-forme Windows, pour spécifier un chemin d'accès dans une chaîne de caractères délimitée par des guillemets, vous devez échapper l'anti-slash (par un anti-slash = `\\`) ou vous pouvez utiliser une notation de type "Unix", avec des slashs (`/`). Par exemple, le chemin `c:\temp\info.txt` peut être écrit "`c:\\temp\\info.txt`" ou "`c:/temp/info.txt`". Si aucun chemin n'est indiqué, c'est le répertoire courant qui est utilisé. Des noms relatifs peuvent être spécifiés en utilisant le caractère `.` (point) pour désigner le répertoire courant et `..` (deux points) pour désigner le répertoire supérieur.

La constante prédéfinie `DIRECTORY_SEPARATOR` donne le caractère de séparation utilisé dans les noms de répertoire pour la plate-forme sur laquelle PHP est installée. La constante prédéfinie `PHP_EOL` donne la séquence de caractères utilisée par la plate-forme pour représenter une nouvelle ligne.

Par ailleurs, plusieurs fonctions possèdent un paramètre (appelé `utiliser_inclusion` dans les syntaxes de cet ouvrage) qui permet de rechercher le fichier dans les répertoires spécifiés par la directive de configuration `include_path`.

### fopen

La fonction `fopen` permet d'ouvrir un fichier.

*Syntaxe*

```
ressource fopen(chaîne nom_fichier, chaîne mode [, booléen
utiliser_inclusion])
```

`nom_fichier`

Chemin d'accès au fichier à ouvrir.

`mode`

Mode d'ouverture du fichier :

`r` = lecture seule

`+` = lecture/écriture

`w` = écriture seule (vide le fichier s'il existe, le crée s'il n'existe pas)

`w+` = lecture/écriture (vide le fichier s'il existe, le crée s'il n'existe pas)

`a` = écriture seule (en ajout - crée le fichier s'il n'existe pas)

`a+` = lecture/écriture (en ajout - crée le fichier s'il n'existe pas)

`x` = écriture seule avec création préalable du fichier (génère une erreur si le fichier existe déjà)

`x+` = lecture/écriture avec création préalable du fichier (génère une erreur si le fichier existe déjà)

Sous Windows, ajouter un `b` pour manipuler les fichiers binaires (cette option, si présente, est ignorée sous Unix) ; utiliser un `t` à la place du `b` spécifie un mode texte (les séquences `\n` sont automatiquement remplacées par un `\r\n`).

`utiliser_inclusion`

Mettre `TRUE` pour rechercher le fichier dans les répertoires spécifiés par la directive de configuration `include_path`.

La fonction `fopen` retourne un pointeur de fichier en cas de succès et `FALSE` en cas d'erreur.

---

Pour des raisons de portabilité, il est recommandé de toujours utiliser l'option `b` lorsque vous ouvrez des fichiers avec `fopen`.

---

## fclose

La fonction `fclose` permet de fermer un fichier.

*Syntaxe*

```
booléen fclose(ressource fichier)
```

`fichier`

Pointeur de fichier préalablement ouvert.

La fonction `fclose` retourne `TRUE` en cas de succès et `FALSE` en cas d'échec.

## fread

La fonction `fread` permet de lire le contenu d'un fichier (dans une chaîne).

*Syntaxe*

```
chaîne fread(ressource fichier, entier longueur)
```

`fichier`

Pointeur de fichier préalablement ouvert.

`longueur`

Nombre d'octets à lire dans le fichier.

La fonction `fread` retourne les données lues ou `FALSE` en cas d'erreur.

## file

La fonction `file` permet de lire le contenu d'un fichier (dans un tableau).

*Syntaxe*

```
tableau file(chaîne nom_fichier [, entier indicateur])
```

`nom_fichier`

Chemin d'accès au fichier à lire.

`indicateur`

Une ou plusieurs des constantes suivantes :

`FILE_USE_INCLUDE_PATH` : rechercher le fichier dans les répertoires spécifiés par la directive de configuration `include_path`.

`FILE_IGNORE_NEW_LINES` : ne pas ajouter la séquence de nouvelle ligne à la fin de chaque ligne du tableau.

`FILE_SKIP_EMPTY_LINES` : ignorer les lignes vides.

### fwrite

La fonction `fwrite` permet d'écrire dans un fichier.

*Syntaxe*

```
entier fwrite(ressource fichier, chaîne données[, entier longueur])
```

`fichier`

Pointeur de fichier préalablement ouvert.

`données`

Données à écrire dans le fichier.

`longueur`

Si précisé, indique le nombre d'octets à écrire (toute la chaîne `données` par défaut).

La fonction `fwrite` retourne le nombre d'octets écrits ou `FALSE` en cas d'erreur.

### file_get_contents

La fonction `file_get_contents` permet de lire le contenu d'un fichier (dans une chaîne).

*Syntaxe*

```
chaîne file_get_contents(chaîne nom_fichier [, booléen utiliser_inclusion])
```

`nom_fichier`

Chemin d'accès au fichier à lire.

`utiliser_inclusion`

Mettre `TRUE` pour rechercher le fichier dans les répertoires spécifiés par la directive de configuration `include_path`.

La fonction `file_get_contents` ouvre le fichier dont le nom est passé en paramètre et le retourne dans une chaîne. La fonction retourne `FALSE` en cas d'erreur.

Utiliser cette fonction se révèle plus performant qu'ouvrir le fichier (`fopen`), le lire (`fread`) et le refermer (`fclose`).

### file_put_contents

La fonction `file_put_contents` permet d'écrire le contenu d'une chaîne dans un fichier. Cette fonction est apparue en version 5.

*Syntaxe*

```
entier file_put_contents(chaîne nom_fichier, chaîne données [, entier mode])
```

nom_fichier

Chemin d'accès au fichier à lire.

données

Données à écrire dans le fichier.

mode

Constante FILE_USE_INCLUDE_PATH pour écrire dans le premier répertoire spécifié dans la directive de configuration include_path. Constante FILE_APPEND pour ajouter les données à la fin du fichier, s'il existe déjà. Spécifier la somme des deux constantes pour combiner les deux options.

La fonction file_put_contents ouvre le fichier dont le nom est passé en paramètre, écrit le contenu de la chaîne passée en paramètre puis ferme le fichier.

Si le fichier n'existe pas, il est créé. Si le fichier existe déjà, son contenu est remplacé, sauf si la constante FILE_APPEND est spécifiée en troisième paramètre, auquel cas les données sont ajoutées à la fin du fichier.

La fonction retourne le nombre d'octets écrits dans le fichier, ou FALSE en cas d'erreur.

## readfile

La fonction readfile lit un fichier et l'envoie directement vers la sortie.

*Syntaxe*

```
entier readfile(chaîne nom_fichier [, booléen utiliser_inclusion])
```

nom_fichier

Chemin d'accès au fichier à lire.

utiliser_inclusion

Mettre TRUE pour rechercher le fichier dans les répertoires spécifiés par la directive de configuration include_path.

La fonction readfile retourne le nombre d'octets lus ou FALSE en cas d'erreur.

## copy

La fonction copy permet de copier un fichier.

*Syntaxe*

```
booléen copy(chaîne source, chaîne destination)
```

source

Emplacement du fichier source.

destination

Emplacement du fichier destination.

La fonction copy effectue la copie et retourne TRUE en cas de succès ou FALSE en cas d'échec.

## unlink

La fonction unlink permet de supprimer un fichier.

*Syntaxe*

```
booléen unlink(chaîne nom_fichier)
```

nom_fichier

Emplacement du fichier à supprimer.

La fonction unlink supprime le fichier et retourne TRUE en cas de succès ou FALSE en cas d'échec.

### rename

La fonction rename permet de renommer un fichier.

*Syntaxe*

```
booléen rename(chaîne ancien_nom, chaîne nouveau_nom)
```

ancien_nom

Emplacement du fichier à renommer.

nouveau_nom

Nouveau nom du fichier.

La fonction rename renomme le fichier et retourne TRUE en cas de succès ou FALSE en cas d'échec.

### file_exists

La fonction file_exists permet de tester l'existence d'un fichier.

*Syntaxe*

```
booléen file_exists(chaîne nom_fichier)
```

nom_fichier

Emplacement du fichier à tester.

La fonction file_exists retourne TRUE si le fichier existe ou FALSE dans le cas contraire.

### filesize

La fonction filesize permet de lire la taille d'un fichier.

*Syntaxe*

```
entier filesize(chaîne nom_fichier)
```

nom_fichier

Emplacement du fichier.

La fonction filesize retourne la taille du fichier ou FALSE en cas d'erreur.

## 2. Exemple d'utilisation

Ces différentes fonctions peuvent être utilisées pour lire ou écrire dans des fichiers sur le serveur.

*Exemple*

```
<?php
// Ouvrir un fichier en écriture.
$f = fopen('info.txt','wb');
// Ecrire dans le fichier.
fwrite($f, 'Olivier HEURTEL');
// Fermer le fichier.
fclose($f);
// Ouvrir un fichier en lecture
$f = fopen('info.txt','rb');
// Lire dans le fichier.
$texte = fread($f, filesize('info.txt'));
// Fermer le fichier.
fclose($f);
```

```
// Afficher les informations lues.
echo $texte,'<br />';
// Plus simple : utiliser file_get_contents.
$texte = file_get_contents('info.txt');
// Afficher les informations lues.
echo $texte;
?>
```

*Résultat*

```
Olivier HEURTEL
Olivier HEURTEL
```

## 3. Les "guillemets magiques"

Comme nous l'avons vu dans la section Gérer les "guillemets magiques" ("magic quotes"), si la directive de configuration `magic_quotes_runtime` est à `on`, les données lues dans les fichiers texte subiront une protection "magic quotes". Cependant, il est très facile de modifier la valeur de la directive `magic_quotes_runtime` en cours de script grâce à la fonction `set_magic_quotes_runtime`.

*Exemple*

```
<?php
// Ecrire une information dans un fichier.
file_put_contents('info.txt',"C'est l'été");
// Afficher la valeur de la directive magic_quotes_runtime.
$mqr = get_magic_quotes_runtime();
echo "<b>magic_quotes_runtime = $mqr</b><br />";
// Lire et afficher le contenu du fichier.
$texte = file_get_contents('info.txt');
echo $texte,'<br />';
// Modifier la valeur de la directive magic_quotes_runtime.
$mqr = ($mqr == 1)?0:1;
set_magic_quotes_runtime($mqr);
echo "<b>magic_quotes_runtime = $mqr</b><br />";
// Lire et afficher le contenu du fichier.
$texte = file_get_contents('info.txt');
echo $texte,'<br />';
?>
```

*Résultat*

```
magic_quotes_runtime = 0
C'est l'été
magic_quotes_runtime = 1
C\'est l\'été
```

Un appel systématique à `set_magic_quotes_runtime(0)` permet de garantir qu'il n'y aura pas de protection "magic quotes" dans les données lues par la suite dans un fichier texte, et donc se conformer à la stratégie proposée dans la section Gérer les "guillemets magiques" ("magic quotes").

# Envoyer un courrier électronique

## 1. Vue d'ensemble

Un site interactif a souvent besoin d'envoyer des messages électroniques aux utilisateurs, par exemple pour confirmer un achat, une inscription ou envoyer une lettre d'information.

La fonction `mail`, proposée par PHP, permet de répondre simplement à ce genre de besoin. Cette fonction est détaillée dans ce chapitre, d'abord, pour envoyer des messages textes (sans pièce jointe), puis pour envoyer des messages au format MIME (*Multipurpose Internet Mail Extensions*).

En complément, PHP propose une bibliothèque puissante, mais plus complexe d'utilisation, pour gérer des messages selon le protocole IMAP (*Internet Message Access Protocol*). Cette bibliothèque ne sera pas abordée dans cet ouvrage car elle n'est pas indispensable pour répondre au besoin évoqué précédemment.

## 2. Envoyer un message texte sans pièce jointe

La fonction `mail` permet d'envoyer un message électronique.

<u>*Syntaxe*</u>

```
booléen mail(chaîne destinataire, chaîne objet, chaîne message[, chaîne
entête])
```

`destinataire`

Adresse e-mail du destinataire. Des destinataires multiples peuvent être indiqués en les séparant par des virgules.

`objet`

Objet du message.

`message`

Texte du message.

`entête`

En-têtes supplémentaires.

La fonction `mail` envoie le message caractérisé par les différents paramètres à un serveur de messagerie défini par les directives de configuration suivantes :

| | | |
|---|---|---|
| Win32 | `SMTP` | Adresse du serveur SMTP auquel envoyer le message. Exemple :<br>`smtp.gmail.com` |
| | `sendmail_from` | Adresse e-mail de l'émetteur. Exemple :<br>`webmaster@monsite.com`<br>Cette directive doit être présente, même vide. |
| Unix | `sendmail_path` | Chemin d'accès vers l'exécutable du serveur de messagerie (peut comporter des paramètres). Exemple :<br>`sendmail -t -i` |

La fonction `mail` retourne `TRUE` si le message a pu être envoyé au serveur (ce qui ne garantit pas que ce dernier a pu l'envoyer avec succès) et `FALSE` dans le cas contraire. Il n'y a aucun moyen de savoir si le message a bien été envoyé

avec succès ; cette vérification doit être réalisée en dehors de PHP.

Le quatrième paramètre permet de spécifier des informations supplémentaires qui seront envoyées dans l'en-tête du message ; des informations multiples doivent être séparées par la séquence `\r\n`.

---

La séquence à utiliser comme séparateur d'en-tête est souvent source de problème (et de débat dans les forums de discussion). Très souvent, la séquence `\n` seule fonctionne ; parfois, la séquence `\r\n` ne fonctionne pas.

---

*Exemple de message simple*

```
<?php
// Destinataire.
$destinataire = 'contact@olivier-heurtel.fr';
// Objet.
$objet = "Inscription à monSite.com";
// Message.
$message =
'Monsieur Heurtel,
Nous vous remercions pour votre inscription sur note site monSite.com.
Nous espérons que ce site répondra à vos attentes.
Le webmaster.';
// Envoi.
$ok = mail($destinataire,$objet,$message);
?>
```

Cet exemple montre que le message peut être directement écrit sur plusieurs lignes ; il est aussi possible de la construire par concaténation.

*Exemple de message plus complexe*

```
<?php
// Deux destinataires séparés par une virgule.
$destinataires = 'olivier@diane.com,xavier@zeus.fr';
// Objet.
$objet = 'Inscription au marathon';
// Message.
$message .= "Olivier, Xavier,\n";
$message .= "Nous vous confirmons votre inscription au\n";
$message .= "marathon le dimanche 6 avril.\n";
$message .= "Le départ sera donné à 9h00.\n\n";
$message .= "Les organisateurs.\n";
// En-têtes supplémentaires.
$entêtes .= "From: \"Inscription\" <contact@marathon.fr>\n";
$entêtes .= "Reply-To: \"Inscription\" <contact@marathon.fr>\n";
$entêtes .= "X-Priority: 1\n";
// Envoi.
$ok = mail($destinataires,$objet,$message,$entêtes);
?>
```

Les en-têtes supplémentaires sont spécifiés sous la forme `mot_clé: valeur`.

Les en-têtes les plus courants sont les suivants :

`From:`

Origine du message sous la forme `["nom en clair"] <adresse e-mail>`. Exemples : `"Olivier HEURTEL" <contact@olivier-heurtel.fr>`,`<contact@olivier-heurtel.fr>`

`To:`

Destinataire(s) (même format que l'en-tête `From`).

`Reply-To:`

Adresse de réponse (même format que l'en-tête `From`).

`X-Priority:`

Priorité du message, de 1 à 4 : 1 = priorité la plus haute ; 2 = priorité haute ; 3 = priorité normale ; 4 = priorité basse

---

Dans le paramètre destinataire, il n'est pas possible de spécifier une adresse sous la forme `["nom en clair"] <adresse e-mail>`. Par contre, il est possible d'indiquer des adresses sous cette forme en les envoyant dans l'en-tête `To:`, en doublon du paramètre `destinataire`.

---

## 3. Envoyer un message au format MIME

### a. Préambule

Dans cette partie, nous allons étudier comment envoyer des messages au format MIME ou plus généralement au format Multipart MIME.

Le format MIME permet d'envoyer un message ayant un autre format que du texte : image, format HTML, etc.

Le format Multipart MIME permet d'envoyer un message composé de plusieurs parties ayant chacune un format différent (du texte plus une image, par exemple), une des "parties" pouvant être une pièce jointe.

L'objectif de cette partie, sans rentrer dans le détail du format MIME (les amateurs peuvent se pencher sur les nombreuses RFC qui traitent du sujet), est de montrer concrètement comment procéder sur deux cas typiques, l'envoi d'un message au format HTML et l'envoi d'un message avec une pièce jointe.

### b. Message au format HTML

Le cas de l'envoi d'un message au format HTML permet d'illustrer l'utilisation du format MIME simple.

*Exemple (source d'un message MIME au format HTML)*

```
From: "Olivier" <olivier@diane.com>
To: "Xavier" <xavier@zeus.fr>
Subject: Bonjour !
Date: Mon, 10 Sep 2001 09:24:13 -0100
Message-ID: <3b9c6a403d9f000b@hermes.diane.com>
MIME-Version: 1.0
Content-Type: text/html; charset=iso-8859-1
Content-Transfer-Encoding: 8bit

<html>
<head><title>Bonjour !</title></head>
<body>
<font color="green">Bonjour !</font>
</body>
</html>
```

Un message MIME simple comporte d'abord les en-têtes standards d'un message, puis trois lignes d'en-têtes supplémentaires (en gras) indiquant que le message est au format MIME, et enfin le corps du message proprement dit.

Les trois lignes d'en-têtes supplémentaires sont les suivantes :

`MIME-Version`

Indique que le message est au format MIME et précise la version.

`Content-Type`

Indique le type MIME du contenu.

`Content-Transfer-Encoding`

Indique le type d'encodage.

Quelques types MIME usuels :

`text/plain`

Texte simple. Le jeu de caractères utilisé peut être spécifié par l'option `charset` (par exemple `iso-8859-1`).

`text/html`

Format HTML. Le jeu de caractères utilisé peut être spécifié par l'option `charset` (par exemple `iso-8859-1`).

`image/jpeg`

Image au format JPEG.

`application/octet-stream`

Données binaires.

Quelques types d'encodage usuels :

`7bit`

Encodage du texte sur 7 bits.

`8bit`

Encodage du texte sur 8 bits (à utiliser pour conserver les caractères accentués).

`base64`

Encodage pour les données binaires (voir la fonction `base64_encode`).

Classiquement, pour les types MIME `text/*`, un type d'encodage `7bit` ou `8bit` est utilisé et le corps du message contient le texte en clair. Il est aussi possible d'utiliser un encodage `base64` et de mettre dans le corps du message des données encodées en conséquence (voir la fonction `base64_encode` plus loin).

Pour les types MIME correspondant à des données binaires (image, son, document PDF ...), un encodage `base64` est utilisé et le corps du message contient les données encodées en conséquence (voir la fonction `base64_encode` plus loin).

Envoyer un message au format HTML avec la fonction `mail` est relativement simple. Il faut :

- placer les lignes adéquates dans l'en-tête ;
- mettre les données HTML dans le corps du message.

*Exemple*

```
<?php
// Destinataires.
$destinataires = 'xavier@zeus.fr';
// Objet.
$objet = 'Bonjour !';
// En-têtes supplémentaires.
$entêtes .= "From: \"Olivier\" <olivier@diane.com>\n";
$entêtes .= "MIME-Version: 1.0\n";
$entêtes .= "Content-Type: text/html; charset=iso-8859-1\n";
$entêtes .= "Content-Transfer-Encoding: 8bit\n";
// Message (HTML).
$message .= "<html>\n";
$message .= "<head><title>Bonjour !</title></head>\n";
$message .= "<body>\n";
$message .= "<font color=\"green\">Bonjour !</font>\n";
$message .= "</body>\n";
$message .= "</html>\n";
// Envoi.
$ok = mail($destinataires,$objet,$message,$entêtes);
?>
```

Si vous utilisez un encodage base64, vous pouvez appeler la fonction base64_encode pour effectuer l'encodage des données.

*Syntaxe*

```
chaîne base64_encode(chaîne données)
```

données

Données à encoder.

Cette fonction retourne les données encodées.

En complément, pour se conformer aux spécifications, il faut découper les données encodées en base64 en morceaux de 76 octets séparés par une séquence \r\n.

Cette opération peut être réalisée très simplement grâce à la fonction chunk_split.

*Syntaxe*

```
chaîne chunk_split (chaîne données [, entier longueur [, chaîne séparateur]])
```

données

Données à découper.

longueur

Longueur des morceaux (76 par défaut).

séparateur

Séparateur des morceaux (\r\n par défaut).

La fonction chunk_split retourne la chaîne découpée.

*Exemple d'utilisation pour l'envoi d'un message*

```
<?php
// Destinataires.
$destinataires = 'xavier@zeus.fr';
// Objet.
$objet = 'Bonjour !';
// En-têtes supplémentaires.
$entêtes  = '';
$entêtes .= "From: \"Olivier\" <olivier@diane.com>\r\n";
$entêtes .= "MIME-Version: 1.0\r\n";
$entêtes .= "Content-Type: text/html; charset=iso-8859-1\r\n";
$entêtes .= "Content-Transfer-Encoding: base64\r\n";
// Message (HTML).
$message .= "<html>\n";
$message .= "<head><title>Bonjour !</title></head>\n";
$message .= "<body>\n";
$message .= "<font color=\"green\">Bonjour !</font>\n";
$message .= "</body>\n";
$message .= "</html>\n";
// Encodage et découpage.
$message = chunk_split(base64_encode($message));
// Envoi.
$ok = mail($destinataires,$objet,$message,$entêtes);
?>
```

### c. Message avec pièce jointe

Le cas de l'envoi d'un message avec pièce jointe permet d'illustrer l'utilisation du format Multipart MIME.

*Exemple (source d'un message avec pièce jointe au format Multipart MIME)*

```
From: "Olivier" <olivier@diane.com>
```

```
To: "Xavier" <xavier@zeus.fr>
Subject: Bonjour !
Date: Mon, 10 Sep 2001 09:24:13 -0100
Message-ID: <3b9c6a403d9f000b@hermes.diane.com>
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=O=L=I=V=I=E=R="

--=O=L=I=V=I=E=R=
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: 8bit

Voir la pièce jointe.

--=O=L=I=V=I=E=R=
Content-Type: application/octet-stream; name="PJ.DOC"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="PJ.DOC"

0M8R4KGxGuEAAA ...
AAAAAAAAAAAAAAAAAAAAAAAAAA==

--=O=L=I=V=I=E=R=--
```

Un message Multipart MIME comporte d'abord les en-têtes standards d'un message, puis deux lignes d'en-têtes supplémentaires (en gras) indiquant que le message est au format Multipart MIME, et enfin les différentes parties du message ; chaque partie possède son propre en-tête indiquant son format (type MIME et encodage).

Dans l'en-tête du message, nous retrouvons les deux lignes `MIME-Version` et `Content-Type`. Pour un message Multipart MIME, `Content-Type` est égal à `multipart/mixed` suivi d'une option `boundary` qui donne la chaîne utilisée pour marquer le début des différentes parties du message (`=O=L=I=V=I=E=R=` sur notre exemple).

Chaque partie du message commence par deux signes "moins" suivis de la chaîne donnée par `boundary` (ce qui donne `--=O=L=I=V=I=E=R=` sur notre exemple).

Dans l'en-tête de chaque partie, nous retrouvons les deux lignes `Content-Type` et `Content-Transfer-Encoding` qui précisent le type de MIME et l'encodage de la partie.

Les données de la partie viennent ensuite.

La fin du message est marquée par deux signes "moins", suivis de la chaîne donnée par `boundary` suivie de deux signes "moins" (ce qui donne `--=O=L=I=V=I=E=R=--` sur notre exemple).

La chaîne donnée par `boundary` doit être choisie avec soin pour éviter d'être confondue avec des données, ce qui conduirait à une mauvaise interprétation du message.

Dans l'en-tête de chaque partie, il est possible d'indiquer une ligne d'en-tête supplémentaire (`Content-Disposition`) qui va suggérer au client de messagerie une manière de présenter les données correspondante à l'utilisateur : les deux valeurs possibles sont `inline` (les données sont présentées directement dans le message) ou `attachment` (les données sont présentées en pièce jointe) ; dans ce cas, un nom de fichier peut être suggéré grâce à l'option `filename`.

*Exemple*

```
Content-Disposition: attachment; filename="PJ.DOC"
```

Pour envoyer un message Multipart MIME avec la fonction `mail`, il faut construire les différentes parties du message dans le paramètre `message`, en respectant la structure indiquée précédemment ; le paramètre `entête` de la fonction `mail` contient les en-têtes standards, y compris l'en-tête indiquant qu'il s'agit d'un message au format Multipart MIME.

Les différentes parties du message doivent être séparées par une ligne vide (`\r\n`). Au sein de chaque partie, il en est de même entre l'en-tête de la partie et les données. Voir la remarque dans le titre Envoyer un courrier électronique - Envoyer un message texte sans pièce jointe au sujet de la séquence `\r\n`.

*Exemple d'envoi d'un message avec pièce jointe*

```
<?php
// Destinataires.
$destinataires = 'xavier@zeus.fr';
// Objet.
$objet = 'Bonjour !';
```

```
// En-têtes supplémentaires.
$entêtes  = '';
// -> origine du message
$entêtes .= "From: \"Olivier\" <olivier@diane.com>\r\n";
// -> message au format Multipart MIME
$entêtes .= "MIME-Version: 1.0\r\n";
$entêtes .= "Content-Type: multipart/mixed; ";
$entêtes .= "boundary=\"=O=L=I=V=I=E=R=\"\r\n";
// Message.
$message  = "";
// -> première partie du message (texte proprement dit)
//    -> en-tête de la partie
$message .= "--=O=L=I=V=I=E=R=\r\n";
$message .= "Content-Type: text/plain; ";
$message .= "charset=iso-8859-1\r\n ";
$message .= "Content-Transfer-Encoding: 8bit\r\n";
$message .= "\r\n"; // ligne vide
//    -> données de la partie
$message .= "Voir la pièce jointe.\r\n";
$message .= "\r\n"; // ligne vide
// -> deuxième partie du message (pièce-jointe)
//    -> en-tête de la partie
$message .= "--=O=L=I=V=I=E=R=\r\n";
$message .= "Content-Type: application/octet-stream; ";
$message .= "name=\"PJ.DOC\"\r\n";
$message .= "Content-Transfer-Encoding: base64\r\n";
$message .= "Content-Disposition: attachment; ";
$message .= "filename=\"pj.doc\"\r\n";
$message .= "\r\n"; // ligne vide
//    -> lecture du fichier correspond à la pièce jointe
//       (s'assurer qu'il n'y a pas d'encodage magic quotes)
set_magic_quotes_runtime(0);
$données = file_get_contents('pj.doc');
//    -> encodage et découpage des données
$données = chunk_split(base64_encode($données));
//    -> données de la partie (intégration dans le message)
$message .= "$données\r\n";
$message .= "\r\n"; // ligne vide
// Délimiteur de fin du message.
$message .= "--=O=L=I=V=I=E=R=--\r\n";
// Envoi du message.
$ok = mail($destinataires,$objet,$message,$entêtes);
?>
```

Plusieurs pièces jointes peuvent être envoyées en répétant, autant de fois que nécessaire, les lignes de code correspondantes.

---

En théorie, la totalité du message pourrait être construite dans l'en-tête (paramètre message égal à une chaîne vide). Dans la pratique, et en règle générale, cela fonctionne sur une plate-forme Linux, mais pas sur une plate-forme Windows.

---

# Manipuler les en-têtes HTTP

La fonction header permet d'envoyer des en-têtes HTTP avec la page HTML.

*Syntaxe simplifiée*

```
header(chaîne en-tête [, booléen remplacer])
```

`en-tête`

Chaîne à envoyer comme en-tête HTTP avec la page HTML.

`remplacer`

Indique si la fonction doit remplacer un en-tête précédemment émis (valeur TRUE, par défaut) ou bien ajouter un nouvel en-tête (valeur FALSE).

Les différents en-têtes HTTP sont décrits dans la RFC 2616.

Par exemple, la fonction header peut être utilisée pour envoyer un en-tête qui interdit la mise en cache de la page par le client ou par un proxy. Ce besoin est assez fréquent dans les scripts PHP qui génèrent du HTML dynamique dont le contenu change en fonction de l'utilisateur.

*Exemple*

```
// HTTP 1.0
header("Pragma: no-cache");
// HTTP 1.1
header("Cache-Control: no-cache, must-revalidate");
```

Dans la suite de cet ouvrage, nous aurons l'occasion d'utiliser la fonction header dans plusieurs situations :

- redirection HTTP (cf. chapitre Gérer les liens et les formulaires avec PHP - Aller sur une autre page) ;
- identification HTTP (cf. chapitre Gérer les sessions - Authentification) ;
- téléchargement ("download") d'un document (cf. chapitre Gérer les liens et les formulaires avec PHP - Échanger un fichier entre le client et le serveur) ;
- affichage d'une image dans une page à l'aide d'un script PHP (cf. chapitre Accéder à une base de données MySQL - Utilisation de l'extension MySQLi).

---

La fonction header doit être appelée avant toute instruction (PHP ou HTML) qui a pour effet de commencer à construire la page HTML (c'est en quelque sorte trop tard pour l'en-tête). Un simple espace, dans le script, ou dans un script inclus (require ou include) provoque une erreur.

---

*Exemple d'appel à la fonction header qui génère une erreur*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
  <?php
  header('location: accueil.htm');
  exit;
  ?>
  </body>
</html>
```

*Résultat*

```
Warning: Cannot modify header information - headers already
```

```
sent by (output started at /app/scripts/info.php:6) in
/app/scripts/saisie.php on line 7
```

Si la fonctionnalité de mise en buffer de la page est activée avec la directive de configuration `output_buffering`, le résultat du script n'est pas envoyé au fur et à mesure mais mis dans un buffer puis envoyé d'un seul coup à la fin (ou par morceaux si le buffer est limité en taille). Dans ce cas, il est possible d'utiliser la fonction `header` sans provoquer d'erreur alors que le script a commencé à construire la page.

PHP propose d'autres fonctions relatives aux en-têtes :

- `headers_list` : liste des en-têtes de la réponse (version 5).
- `header_sent` : permet de tester si les en-têtes ont déjà été envoyés.
- `get_headers` : liste des en-têtes renvoyés par un serveur, pour une URL donnée (version 5).

# Fonctions

## 1. Introduction

À l'instar des différents langages de développement, PHP offre la possibilité de définir ses propres fonctions (appelées fonctions « utilisateur ») avec tous les avantages associés (modularité, capitalisation...). Une fonction est un ensemble d'instructions identifiées par un nom, dont l'exécution retourne une valeur et dont l'appel peut être utilisé comme opérande dans une expression. Une procédure est un ensemble d'instructions identifiées par un nom qui peut être appelé comme une instruction.

## 2. Déclaration et appel

Le mot clé `function` permet d'introduire la définition d'une fonction.

*Syntaxe*

```
function nom_fonction([paramètres]) {
  instructions;
}
```

`nom_fonction`

Nom de la fonction (doit respecter les règles de nommage présentées dans le chapitre Introduction à PHP - Structure de base d'une page PHP).

`paramètre`

Paramètres éventuels de la fonction exprimés sous forme d'une liste de variables (cf. Paramètres) : `$paramètre1`, `$paramètre2`, ...

`instructions`

Ensemble des instructions qui composent la fonction.

Le nom de la fonction ne doit pas être un mot réservé PHP (nom de fonction, d'instruction) ni être égal au nom d'une autre fonction préalablement définie.

Une fonction utilisateur s'appelle comme une fonction native de PHP : dans une affectation, dans une comparaison, etc.

Si la fonction retourne une valeur, il est possible d'utiliser l'instruction `return` pour définir la valeur de retour de la fonction.

*Syntaxe*

```
return [expression];
```

`expression`

Expression dont le résultat constitue la valeur de retour de la fonction (`NULL` par défaut).

Le résultat d'une fonction peut être de n'importe quel type (chaîne, nombre, tableau, etc.).

L'instruction `return` stoppe l'exécution de la fonction et retourne le résultat de `expression` à l'appelant. Si plusieurs instructions `return` sont présentes dans la fonction, c'est la première rencontrée dans le déroulement des instructions qui définit la valeur de retour et provoque l'interruption de la fonction. Si la fonction ne comporte aucune instruction `return` (ou si aucune instruction `return` n'est exécutée), la valeur de retour de la fonction est `NULL`.

*Exemple*

```
<?php
// Fonction sans paramètre qui affiche "Bonjour !"
// pas de valeur de retour
function afficher_bonjour() {
  echo 'Bonjour !<br />';
}
```

```
// Fonction avec 2 paramètres qui retourne le produit
// des deux paramètres.
function produit($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
// Appel de la fonction afficher_bonjour
afficher_bonjour();
// Utilisations de la fonction produit :
// - dans une affectation
$résultat = produit(2,4);
echo "2 x 4 = $résultat<br />";
// - dans une comparaison
if (produit(10,12) > 100) {
  echo '10 x 12 est supérieur à 100.<br />';
}
?>
```

*Résultat*

```
Bonjour !
2 x 4 = 8
10 x 12 est supérieur à 100.
```

Dans le langage PHP, il n'existe pas à proprement parler de procédure. Pour définir quelque chose d'équivalent à une procédure, il suffit de définir une fonction qui ne retourne par de valeur et d'appeler la fonction comme si c'était une instruction (comme la fonction `afficher_bonjour` par exemple).

Il est possible d'utiliser une fonction avant de la définir. Il n'y a donc aucun problème pour définir des fonctions qui s'appellent entre elles.

Une fonction est utilisable uniquement dans le script où elle est définie. Pour pouvoir l'utiliser dans plusieurs scripts, le mieux est de la définir dans un fichier inclus partout où la fonction est nécessaire.

PHP permet de stocker un nom de fonction dans une variable et d'appeler la variable dans une instruction, comme si c'était une fonction, avec la notation `$variable()`. Sur une telle écriture, PHP remplace la variable par sa valeur et cherche à exécuter la fonction correspondante (qui doit, bien entendu, exister). Cette fonctionnalité est appelée "fonction variable".

*Exemple*

```
<?php
// Fonction qui effectue un produit.
function produit($valeur1,$valeur2) {
  return $valeur1 * $valeur2;
}
// Fonction qui effectue une somme.
function somme($valeur1,$valeur2) {
  return $valeur1 + $valeur2;
}
// Fonction qui effectue un calcul, le nom du calcul
//('somme' ou 'produit') étant passé en paramètre.
function calculer($opération,$valeur1,$valeur2) {
  // $opération contient le nom de la fonction
  // a exécuter => appel $opération().
  return $opération($valeur1,$valeur2);
}
// Utilisation de la fonction calculer.
echo '3 + 5 = ',calculer('somme',3,5).'<br />';
echo '3 x 5 = ',calculer('produit',3,5).'<br />';

?>
```

*Résultat*

```
3 + 5 = 8
3 x 5 = 15
```

## 3. Paramètres

Les paramètres (aussi appelés arguments) éventuels d'une fonction sont définis sous la forme d'une liste de variables.

Dans ce paragraphe, nous allons étudier les possibilités suivantes :

- définir une valeur par défaut pour un paramètre ;
- passer un paramètre par référence ;
- utiliser une liste variable de paramètres.

### a. Valeur par défaut

Il est possible d'indiquer qu'un paramètre possède une valeur par défaut grâce à la syntaxe suivante :

```
$paramètre = expression_littérale
```

La valeur par défaut d'un paramètre doit être une expression littérale et ne peut être ni une variable, ni une fonction, ni une expression composée. Les constantes sont autorisées.

La valeur par défaut est utilisée comme valeur d'un paramètre lorsque la fonction est appelée sans mentionner de valeur pour le paramètre en question.

*Exemple*

```
<?php
// Définition d'une constante
define('UN',1);
// Définition de la fonction produit avec des valeurs
// par défaut pour les paramètres (dont une constante
// pour le premier paramètre).
function produit($valeur1=UN,$valeur2=2) {
  return $valeur1 * $valeur2;
}
// Appels
// - sans paramètre
echo 'produit() = ',produit(),'<br />';
// - avec un seul paramètre = forcément le premier
echo 'produit(3) = ',produit(3),'<br />';
?>
```

*Résultat*

```
produit() = 2
produit(3) = 6
```

Ne pas donner de valeur à un paramètre ayant une valeur par défaut n'est possible qu'en partant de la droite. Passer une valeur "vide" (chaîne vide ou NULL) ne résout pas le problème car la valeur en question est convertie par PHP dans le type adéquat (0 sur l'exemple précédent).

Passer un nombre insuffisant de paramètres et ne pas avoir de valeur par défaut génère une erreur. Passer trop de paramètres ne génère pas d'erreur ; les paramètres en trop sont ignorés.

### b. Passage par référence

Par défaut, le passage des paramètres s'effectue par valeur : c'est une copie de la valeur qui est passée à la fonction. En conséquence, la modification des paramètres à l'intérieur de la fonction n'a aucun effet sur les valeurs dans le script appelant.

En cas de besoin, il est possible d'avoir un passage par référence en utilisant l'opérateur de référence & (cf. chapitre Introduction à PHP - Les bases du langage PHP) devant le nom du paramètre dans la définition de la fonction. Avec une telle définition, c'est une référence vers la variable (et non une copie) qui est passée à la fonction ; cette dernière travaille directement sur la variable du script appelant.

*Exemple*

```
<?php
// Définition d'une fonction qui prend deux paramètres :
// un passé par valeur et l'autre par référence.
function une_fonction($par_valeur,&$par_référence) {
  // Incrémentation des deux paramètres.
  $par_valeur++;
  $par_référence++;
  // Affichage des deux paramètres à l'intérieur
  // de la fonction.
  echo "\$par_valeur = $par_valeur<br />";
  echo "\$par_référence = $par_référence<br />";
}

// Initialisation de deux variables.
$x = 1;
$y = 10;
// Affichage des variables avant l'appel à la fonction.
echo "\$x avant appel = $x<br />";
echo "\$y avant appel = $y<br />";
// Appel de la fonction en utilisant les deux variables comme
// valeur des paramètres.
une_fonction($x,$y);
// Affichage des variables après l'appel à la fonction.
echo "\$x après appel = $x<br />";
echo "\$y après appel = $y<br />";
?>
```

*Résultat*

```
$x avant appel = 1
$y avant appel = 10
$par_valeur = 2
$par_référence = 11
$x après appel = 1
$y après appel = 11
```

Avec une telle définition, il n'est plus possible de passer une constante ou une expression comme valeur du paramètre.

### c. Liste variable de paramètres

À l'intérieur d'une fonction, il est possible d'utiliser les trois fonctions PHP suivantes :

| Fonction | Rôle |
|---|---|
| `func_num_args` | Donne le nombre de paramètres passés à la fonction. |
| `func_get_args` | Retourne la liste des paramètres passés à la fonction (dans un tableau). |
| `func_get_arg` | Retourne la valeur d'un paramètre dont le numéro est précisé. |

*Syntaxe*

```
entier func_num_args()
tableau func_get_args()
mixte func_get_arg(entier numéro)
```

`numéro`

Numéro du paramètre demandé (0 = premier paramètre).

À l'aide de ces fonctions natives, il est très simple d'écrire une fonction qui accepte un nombre variable de paramètres. Les principes sont les suivants :

- déclarer la fonction sans paramètre ;
- récupérer, dans le corps de la fonction, les paramètres avec les fonctions `func_get_args` ou `func_get_arg` et les utiliser (typiquement dans une boucle).

Dans la pratique, rien n'interdit aux paramètres d'être de type différent.

Par ailleurs, il est possible de déclarer explicitement les premiers paramètres et d'accepter ensuite une liste variable de paramètres complémentaires. Dans ce cas, les paramètres explicitement déclarés sont repris dans le comptage et dans la liste des paramètres. Il convient donc de les éliminer soi-même du traitement.

*Exemple*

```
<?php
// Fonction qui accepte un premier paramètre par référence
// et qui y stocke le produit de tous les autres paramètres.
function produit(&$résultat) {
  switch (func_num_args()) {
    case 1:
      // Un seul paramètre (la variable pour le résultat)
      // Retourner 0 (choix arbitraire).
      $résultat = 0;
      break;
    default:
      // Récupérer les paramètres dans un tableau
      // et supprimer le premier élément (le premier
      // paramètre).
      $paramètres = func_get_args();
      unset($paramètres[0]);
      // Initialiser le résultat à 1
      $résultat = 1;
      // Faire une boucle sur le tableau des paramètres
      // pour multiplier le résultat par le paramètre.
      foreach($paramètres as $paramètre) {
        $résultat *= $paramètre;
      }
      break;
  }
}
// appels
produit($résultat);
echo 'produit($résultat) => ',$résultat,'<br />';
produit($résultat,1,2,3);
echo 'produit($résultat,1,2,3) => ',$résultat,'<br />';
?>
```

*Résultat*

```
produit($résultat) => 0
produit($résultat,1,2,3) => 6
```

## 4. Considérations sur les variables utilisées dans les fonctions

### a. Variable locale - globale

Les variables utilisées à l'intérieur d'une fonction sont locales : elles sont non définies en dehors de la fonction et initialisées à chaque appel de la fonction. Il en est de même des paramètres de la fonction.

Réciproquement, une variable définie en dehors de la fonction (dans le script appelant) n'est pas définie à l'intérieur de la fonction.

PHP propose une notion de variable globale pour accéder dans une fonction aux variables définies dans le contexte du script appelant.

Pour cela, à l'intérieur de la fonction, il faut déclarer les variables globales que la fonction utilise avec le mot clé `global` ou utiliser le tableau associatif prédéfini `$GLOBALS`.

*Syntaxe*

```
global $variable[, ...];
```

`$variable`

Variable de la fonction qui a une portée globale. Plusieurs variables peuvent être mentionnées, en les séparant par une virgule.

Le tableau prédéfini `$GLOBALS` est un tableau associatif. Dans ce tableau associatif, la clé est égale au nom de la variable globale (sans le $) et la valeur associée est égale à la valeur de la variable globale.

---

Le tableau associatif `$GLOBALS` est un tableau "superglobal". Un tableau superglobal est automatiquement disponible dans tous les environnements d'exécution, sans avoir à le déclarer global. Dans ce livre, nous utiliserons d'autres tableaux superglobaux.

---

*Exemple*

```
<?php
// Objectif : écrire une fonction qui effectue le produit
//            des variables $x et $y et qui stocke le résultat
//            dans la variable $z.
// Initialisation de deux variables dans le script appelant.
$x = 2;
$y = 5;
echo '<b>Cas 1 : pas d\'utilisation des variables ',
     'globales</b><br />';
function produit1() {
  // $x et $y sont vides à l'intérieur de la fonction.
  echo "\$x = $x<br />";
  echo "\$y = $y<br />";
  $z = 0 + $x * $y;
}
produit1();
// $z est vide dans le script principal.
echo "\$z = $z<br />";
// Résolution du problème en utilisant des variables globales :
// - avec le mot clé global pour $x et $y
// - avec le tableau $GLOBALS pour $z
echo '<b>Cas 2 : utilisation des variables globales</b><br />';
function produit2() {
  global $x, $y;
  echo "\$x = $x<br />";
  echo "\$y = $y<br />";
  $GLOBALS['z'] = 0 + $x * $y;
}
produit2();
echo "\$z = $z<br />";
?>
```

*Résultat*

```
Cas 1 : pas d'utilisation des variables globales
$x =
$y =
$z =
Cas 2 : utilisation des variables globales
$x = 2
$y = 5
$z = 10
```

---

Un paramètre d'une fonction se comporte comme une variable locale à la fonction, sauf s'il est passé par référence, auquel cas il est équivalent à une variable globale.

---

### b. Variable statique

Par défaut, les variables locales d'une fonction sont réinitialisées à chaque appel de la fonction.

Le mot clé `static` permet de définir des variables locales statiques qui ont pour propriété de conserver leur valeur d'un appel à l'autre de la fonction, pendant la durée du script.

*Syntaxe*

```
static $variable = expression_littérale[, ...];
```

`$variable`

Variable concernée.

`expression_littérale`

Valeur initiale affectée à la variable lors du premier appel à la fonction à l'intérieur du script. Seules les expressions littérales et les constantes sont acceptées ; les expressions composées ou les fonctions ne sont pas autorisées.

*Exemple*

```
<?php
// Définition d'une fonction
function variable_statique() {
  // Initialisation d'une variable statique.
  static $variable_statique = 0;
  // Initialisation d'une autre variable.
  $autre_variable = 0;
  // Affichage des deux variables.
  echo "\$variable_statique = $variable_statique <br />";
  echo "\$autre_variable = $autre_variable<br />";
  // Incrémentation des deux variables.
  $variable_statique++;
  $autre_variable++;
}
// Premier appel de la fonction.
echo '<b>Premier appel de la fonction :</b><br />';
variable_statique();
// ...
// Deuxième appel de la fonction.
echo '<b>Deuxième appel de la fonction :</b><br />';
variable_statique();
//...
// Troisième appel de la fonction.
echo '<b>Troisième appel de la fonction :</b><br />';
variable_statique();

?>
```

*Résultat*

```
Premier appel de la fonction :
$variable_statique = 0
$autre_variable = 0
Deuxième appel de la fonction :
$variable_statique = 1
$autre_variable = 0
Troisième appel de la fonction :
$variable_statique = 2
$autre_variable = 0
```



La valeur n'est conservée que pendant la durée du script : lorsque ce dernier se termine, la valeur est perdue et au prochain appel du script, la variable statique est réinitialisée.

## 5. Les constantes et les fonctions

Dans le chapitre Introduction à PHP, nous avons vu que la portée des constantes est le script dans lequel elles sont définies.

À la différence des variables, cette portée s'étend aux fonctions utilisées dans le script : une constante peut être utilisée à l'intérieur de la fonction sans qu'elle soit déclarée globale. Réciproquement, une constante définie dans une fonction peut être utilisée dans un script, après appel de la fonction.

*Exemple*

```
<?php
// Définition d'une constante dans le script.
define(CONSTANTE_SCRIPT,'constante script');
// Définition d'une fonction.
function constante() {
  // Qui définit une constante.
  define(CONSTANTE_FONCTION,"constante fonction");
  // Et qui affiche une constante du script appelant.
  echo 'Dans la fonction, CONSTANTE_SCRIPT = ',
        CONSTANTE_SCRIPT,'<br />';
}
// Appel de la fonction.
constante();
// Affichage de la constante définie dans la fonction.
echo 'Dans le script, CONSTANTE_FONCTION = ',
   CONSTANTE_FONCTION,'<br />';
?>
```

*Résultat*

```
Dans la fonction, CONSTANTE_SCRIPT = constante script
Dans le script, CONSTANTE_FONCTION = constante fonction
```

## 6. Récursivité

À l'instar de nombreux langages, PHP autorise la récursivité, c'est-à-dire la possibilité pour une fonction de s'appeler elle-même.

Pour illustrer cette possibilité, nous allons écrire une fonction générique qui permet d'afficher le contenu d'un tableau, éventuellement multidimensionnel.

*Source*

```
<?php
function  afficher_tableau($tableau,$titre="",$niveau=0) {
  // Paramètres
  //    - $tableau = tableau dont il faut afficher le contenu
  //    - $titre = titre à afficher au dessus du contenu
  //    - $niveau = niveau d'affichage
  // S'il y a un titre, l'afficher.
  if ($titre != "") {
    echo "<br /><b>$titre</b><br />";
  }
  // Tester s'il y a des données.
  if (isset($tableau)) { // il y a des données
    // Parcourir le tableau passé en paramètre.
    reset ($tableau);
    foreach ($tableau as $cle => $valeur) {
      // Afficher la clé (avec indentation fonction
      // du niveau).
      echo
        str_pad('',12*$niveau, ' '),
        htmlentities($cle),' = ';
      // Afficher la valeur
      if (is_array($valeur)) { // c'est un tableau ...
```

```
        // mettre une balise <br />
        echo '<br />';
        // Et appeler récursivement afficher_tableau pour
        // afficher le tableau en question (sans titre et
        // au niveau suivant pour l'indentation)
        afficher_tableau($valeur,'',$niveau+1);
      } else { // c'est une valeur scalaire
        // Afficher la valeur.
        echo htmlentities($valeur),'<br />';
      }
    }
  } else { // pas de données
    // Mettre une simple balise <br />
    echo '<br />';
  }
}
// Afficher un tableau de couleurs.
$couleurs = array('Bleu','Blanc','Rouge');
afficher_tableau($couleurs,'Couleurs');
// Afficher un tableau à deux dimensions (pays/ville).
$pays = array('France' => array('Paris','Lyon','Nantes'),
              'Italie' => array('Rome','Venise'));
afficher_tableau($pays,'Pays/Villes');
?>
```

*Résultat*

```
Couleurs
0 = Bleu
1 = Blanc
2 = Rouge

Pays/Villes
France =
  0 = Paris
  1 = Lyon
  2 = Nantes
Italie =
  0 = Rome
  1 = Venise
```

Pour être rigoureux, il faudrait tester que la variable passée initialement en premier paramètre est bien un tableau.

Nous utiliserons cette fonction à plusieurs reprises dans cet ouvrage, partant de l'hypothèse qu'elle fait partie d'un jeu de fonctions génériques définies dans un fichier qui est inclus dans le script en cas de besoin.

# Classes

## 1. Concept

PHP propose des fonctionnalités classiques de programmation orientée objet :

- définition de classe ;
- utilisation de méthodes constructeur et destructeur ;
- notions d'attribut ou de méthode public, privé, protégé ;
- héritage ;
- notions de classe ou méthode abstraite, de classe ou méthode finale, d'interface, d'attribut ou méthode statique (de classe) ;
- exceptions.

Une classe est un type composite regroupant des variables (appelées attributs de la classe) et des fonctions (appelées méthodes de la classe). En soi, une classe ne contient pas de données ; c'est juste un modèle, une définition.

À partir de la classe, il est possible de définir ("instancier") des objets qui ont la structure de la classe et qui, eux, contiennent des données.

Dans cette partie, nous allons présenter les fonctionnalités de base les plus couramment utilisées : c'est une introduction pratique aux fonctionnalités objet de PHP. Pour plus d'informations, reportez-vous à la documentation de PHP.

## 2. Définir une classe

Le mot clé `class` permet d'introduire la définition d'une classe.

*Syntaxe*

```
class nom_classe {
  // définition des attributs
  [
  public | private | protected $attribut [= littéral];
  ...
  ]
  // définition des méthodes
  [
  [public | private | protected] function méthode() {
    ...
  }
  ...
  ]
}
```

`nom_classe`

Nom de la classe (doit respecter les règles de nommage présentées dans le chapitre Introduction à MySQL).

`$attribut`

Nom d'une variable correspondant à un attribut de la classe.

`littéral`

Valeur initiale de l'attribut. Seules les expressions littérales et les constantes sont acceptées ; les expressions

composées ou les fonctions ne sont pas autorisées.

`méthode`

Définition d'une fonction correspondant à une des méthodes de la classe.

La visibilité des attributs et des méthodes est définie par un des mots clés suivants :

`public`

L'attribut ou la méthode sont publics et l'on peut y accéder de l'extérieur de la classe.

`private`

L'attribut ou la méthode sont privés et l'on ne peut y accéder que de l'intérieur de la classe.

`protected`

L'attribut ou la méthode sont protégés et l'on ne peut y accéder que de l'intérieur de la classe ou des classes dérivées de la classe (voir la notion d'héritage).

Par défaut, une méthode est publique. Par contre, la visibilité de l'attribut doit être spécifiée.

Les méthodes sont définies comme des fonctions utilisateur classiques (avec paramètres, instruction `return`, etc.), mais, à l'intérieur de la classe. Une méthode d'une classe peut avoir le même nom qu'une fonction utilisateur ou qu'une autre méthode d'une autre classe.

Une des méthodes peut porter le même nom que la classe dans laquelle elle est définie. Cette méthode particulière, appelée méthode **constructeur**, est automatiquement appelée à la création ("instanciation") d'un nouvel objet. Généralement, cette méthode est utilisée pour initialiser des attributs qui ne peuvent l'être par une simple expression littérale. La méthode constructeur d'une classe peut être nommée de manière unifiée `__construct`. Cette méthode de nommage est conseillée car elle facilite l'appel de la méthode constructeur d'une classe parent dans une classe dérivée (voir la notion d'héritage). En premier PHP recherche toujours une méthode nommée `__construct` ; s'il n'en trouve pas il recherche une méthode portant le même nom que la classe.

En complément, il est possible de spécifier une méthode **destructeur** nommée `__destruct` (pas de paramètre). Cette méthode destructeur est automatiquement appelée lorsque la dernière référence à un objet est supprimée. Cette méthode peut être utilisée pour libérer des ressources utilisées par l'objet (fichier, connexion à une base de données, etc.).

Enfin, il est possible de spécifier une méthode **tostring** nommée `__toString` (pas de paramètre) qui permet de convertir un objet en chaîne. Cette méthode est automatiquement appelée à chaque fois que l'objet est utilisé dans un contexte où PHP attend une chaîne (par exemple dans un `echo`). Cette méthode doit retourner une chaîne dans laquelle vous pouvez intégrer les informations que vous souhaitez sur l'objet.

À l'intérieur des méthodes, l'objet courant (ou instance courante) peut être référencé par la variable `$this` ; pour accéder aux attributs ou aux méthodes, il suffit d'utiliser l'opérateur `->` suivi du nom de l'attribut ou du nom de la méthode, derrière le nom de la variable `$this`.

*Syntaxe*

```
$this->attribut
$this->méthode([valeur[, ...]])
```

Bien noter que le signe `$` porte sur le nom de la variable `this`, pas sur le nom de l'attribut.

*Exemple*

```
<?php
// Définition d'une classe destinée à stocker des informations
// sur un utilisateur.
class utilisateur {
  // Définition des attributs.
  public  $nom; // nom de l'utilisateur
  public  $prénom; // prénom de l'utilisateur
  public  $langue = 'fr'; // langue de l'utilisateur
                          // français par défaut
  private $timestamp; // date/heure de création (privé)
  // Définition des méthodes :
  // - méthode constructeur
  public function __construct($prénom,$nom) {
```

```
    // Initialiser le nom et le prénom
    // avec les valeurs passées en paramètre.
    $this->prénom = $prénom;
    $this->nom = $nom;
    // Initialiser le timestamp avec la fonction time().
    $this->timestamp = time();
  }
  // - méthode destructeur
  public function __destruct() {
    // Se contente d'afficher un message.
    echo "<p><b>Suppression de $this->nom</b></p>";
  }
  // - méthode de conversion de l'objet en chaîne
  public function __toString() {
    // Retourne juste le nom et le prénom.
    return "__toString = $this->nom - $this->prénom";
  }
  // - méthode qui modifie la langue de l'utilisateur.
  public function langue($langue) {
    $this->langue = $langue;
  }
  // - méthode (privée) qui met en forme la date/heure
  //   de création de l'utilisateur.
  private function miseEnFormeTimestamp() {
    setlocale(LC_TIME,$this->langue);
    return strftime('%c', $this->timestamp);
  }
  // - méthode qui donne des informations sur l'utilisateur
  public function informations() {
    $création = $this->miseEnFormeTimestamp();
    return "$this->prénom $this->nom - $création";
  }
}
?>
```

Une classe est utilisable uniquement dans le script où elle est définie. Pour pouvoir l'utiliser dans plusieurs scripts, le mieux est de la définir dans un fichier qui est inclus partout où la classe est utilisée.

## 3. Instancier une classe

Instancier une classe signifie créer un objet basé sur la définition de la classe. Dans une certaine mesure, cela revient à définir une variable ayant comme "type" la classe.

L'instanciation s'effectue grâce à l'opérateur new.

*Syntaxe*

```
$nom_objet =  new nom_classe[(valeur[, ...])]
```

`$nom_objet`

Variable destinée à stocker l'objet.

`nom_classe`

Nom de la classe qui sert de "modèle" à l'objet.

`valeur`

Paramètre éventuel passé à la méthode constructeur de la classe appelée lors de la création de l'objet.

Après création de l'objet, les attributs et méthodes publics de l'objet peuvent être atteints avec l'opérateur ->, sur la variable `$nom_objet`, comme avec la variable `$this`.

Il n'y a pas de protection des attributs publics des objets ; ils peuvent être manipulés directement.

*Exemple*

```
<?php
// Inclusion du fichier qui contient la définition de la
// classe utilisateur présentée précédemment.
include('classes.inc');
// Instanciation d'un objet.
$moi = new utilisateur('Olivier','Heurtel');
// La variable $moi contient maintenant un objet basé sur la
// classe utilisateur. Les méthodes sont accessibles par
// l'opérateur ->.
// Utilisation des méthodes de l'objet.
echo "{$moi->informations()} <br/>";
$moi->langue('us');  // modification de la langue
echo "{$moi->informations()} <br/>";
// Modification et lecture directe d'un attribut public
$moi->nom = strtoupper($moi->nom);
echo "$moi->nom <br />";
// Affichage direct de l'objet => utilisation de __toString
echo "$moi <br />";
?>
```

*Résultat*

```
Olivier Heurtel - 07/12/2007 21:09:28
Olivier Heurtel - 12/7/2007 09:09:28 PM
HEURTEL
__toString = HEURTEL - Olivier
Suppression de HEURTEL
```

Comme le montre cet exemple, les variables objet peuvent être substituées comme les autres (cf. chapitre Introduction à PHP - Les bases du langage PHP) dans les chaînes de caractères délimitées par des guillemets doubles. En cas de besoin, les accolades peuvent être utilisées pour délimiter la variable à l'intérieur de la chaîne.

Les règles de portée et de durée de vie des variables s'appliquent aux objets (cf. chapitre Introduction à PHP - Les bases du langage PHP).

## 4. Héritage

Il est possible de définir une nouvelle classe qui hérite d'une classe existante avec le mot clé `extends`.

*Syntaxe*

```
class nom_classe extends nom_classe_de_base{
  // Définition des attributs supplémentaires.
  [
  public | private | protected $attribut [= littéral];
  ...
  ]
  // Définition des méthodes supplémentaires.
  [
  [public | private | protected] function méthode() {
    ...
  }
  ...
  ]
}
```

La signification des différents éléments est la même que pour la définition d'une classe.

La nouvelle classe créée est appelée classe fille et la classe de base (qui a servi de "moule" à cette création) est nommée classe mère.

La nouvelle classe possède implicitement les attributs et méthodes de la classe de base et peut en définir de nouveaux, notamment une méthode constructeur nommée `__construct`. Si la classe fille n'a pas de méthode constructeur, c'est la méthode constructeur de la classe mère qui est appelée lors de l'instanciation d'un objet sur la classe fille.

Quand la méthode constructeur existe dans la classe fille, il n'y a pas d'appel automatique à la méthode constructeur de la classe mère : si nécessaire, il faut l'appeler explicitement en utilisant `parent::__construct()`.

*Exemple*

```
<?php
// Définition d'une classe de base.
class utilisateur {
  // Définition des attributs.
  public $nom; // nom de l'utilisateur
  public $prénom; // prénom de l'utilisateur
  // Définition des méthodes :
  // - méthode constructeur
  public function __construct($prénom,$nom) {
    // Initialiser le nom et le prénom
    // avec les valeurs passées en paramètre
    $this->prénom = $prénom;
    $this->nom = $nom;
  }
  // - méthode qui donne les informations sur l'utilisateur
  public function informations() {
    return "$this->prénom $this->nom";
  }
}
// Définition d'une classe qui hérite de la première
class utilisateur_couleur extends utilisateur{
  // Définition des attributs complémentaires.
  public $couleurs; // couleurs préférées de l'utilisateur
  // Définition des méthodes complémentaires
  // - méthode constructeur
  public function __construct($prénom,$couleurs) {
    // Appel au constructeur de la classe mère
    // pour la première partie de l'initialisation.
    parent::__construct ($prénom,'X');
    // Initialisation spécifique complémentaire.
    $this->couleurs = explode(',',$couleurs);
  }
  // - liste des couleurs préférées de l'utilisateur
  public function couleurs() {
    return implode(',',$this->couleurs);
  }
}
// Instanciation d'un objet sur la classe fille.
$moi = new utilisateur_couleur('Olivier','bleu,blanc,rouge');
// Utilisation des méthodes :
// - de la classe mère
echo "{$moi->informations()}<br />"; // existe par héritage
// - de la classe fille
echo "{$moi->couleurs()}<br />"; // existe dans la classe
?>
```

*Résultat*

```
Olivier X
bleu,blanc,rouge
```

---

Dans une classe fille, il est possible de redéclarer une méthode ou un attribut qui existe dans la classe mère. Dans ce cas, l'opérateur `parent::` peut être utilisé pour faire référence explicitement aux attributs (`parent::attribut`) ou méthodes (`parent::méthode()`) de la classe mère et lever l'ambiguïté du nommage (cet opérateur peut être utilisé même s'il n'y a pas d'ambiguïté).

---

## 5. Autres fonctionnalités sur les classes

### a. Classe ou méthode abstraite

Une classe abstraite est une classe qui ne peut pas être instanciée (pas de création d'objet sur la classe). Par contre une telle classe peut servir de base à la définition d'une classe fille qui pourra être instanciée (sauf si elle est elle-même abstraite).

Une méthode abstraite est une méthode qui est définie dans une classe (elle-même obligatoirement abstraite) mais pas implémentée (le code de la méthode n'est pas présent). Une telle méthode pourra être implémentée dans une classe fille. Une méthode abstraite ne peut pas être privée.

En terme de syntaxe, il suffit de mettre le mot clé `abstract` devant la définition de la classe ou de la méthode.

Une classe fille qui n'implémente pas toutes les méthodes abstraites de la classe mère est implicitement abstraite (même si le mot clé `abstract` n'est pas présent).

*Exemple*

```
<?php
// Définition d'une classe abstraite.abstract class classeMère {
  // Attribut protégé.
  protected $x;
  // Deux méthodes pour accéder à l'attribut protégé :
  // - pour lire
  public function get() {
    return "GET = $this->x";
  }
  // - pour écrire
  //   > méthode abstraite
  abstract public function put($valeur);
}
// Définition d'une classe fille qui hérite de la classe mère.
class classeFille extends classeMère {
  // Implémentation de la méthode d'écriture.
  public function put($valeur) {
    $this->x = $valeur;
  }
}
// Utilisation de la classe fille.
$objet = new classeFille();
$objet->put(123);
echo "{$objet->get()} <br />";
?>
```

*Résultat*

```
GET = 123
```

### b. Classe ou méthode finale

On ne peut pas hériter d'une classe finale.

Une méthode finale ne peut pas être redéfinie dans une classe fille.

En termes de syntaxe, il suffit de mettre le mot clé `final` devant la définition de la classe ou de la méthode.

Tenter d'hériter d'une classe finale génère une erreur :

```
Fatal error: Class classeFille may not inherit from final
class (classeMère) in \app\scripts\index.php on line 10
```

Tenter de redéfinir une méthode finale dans une classe dérivée génère une erreur :

```
Fatal error: Cannot override final method
classeMère::méthodeFinale() in \app\scripts\index.php on line 14
```

### c. Interface

Une interface est une classe qui ne contient que des spécifications de méthodes sans implémentations. Une interface ne contient pas d'attributs non plus.

D'autres classes peuvent être ensuite définies et implémenter une ou plusieurs interfaces, c'est-à-dire implémenter les méthodes d'une ou plusieurs interfaces.

*Syntaxe de définition d'une interface*

```
interface nom_interface {
  // Définition des méthodes.
  [public] function méthode();
  ...
}
```

`nom_interface`

Nom de l'interface (doit respecter les règles de nommage présentées dans le chapitre Introduction à PHP - Structure de base d'une page PHP).

`méthode`

Spécification d'une méthode de l'interface.

Les méthodes d'une interface sont forcément publiques ; le mot clé `public` peut être omis.

Il est possible de définir une nouvelle classe qui implémente une ou plusieurs interfaces avec le mot clé `implements`.

*Syntaxe*

```
class nom_classe implements nom_interface[,...] {
...
}
```

Une telle classe doit redéfinir les différentes méthodes des interfaces qu'elle implémente. Si l'une des méthodes des interfaces n'est pas implémentée, la classe doit être déclarée abstraite.

*Exemple*

```
<?php
// Définition de deux interfaces.
interface lecture {
  function get();
}
interface écriture {
  function put($valeur);
}
// Définition d'une classe qui implémente les deux interfaces.
class uneClasse implements lecture,écriture {
  // Définition d'un attribut quelconque.
  private $x;
  // Implémentation de la méthode de lecture.
  public function get() {
    return $this->x;
  }
  // Implémentation de la méthode d'écriture.
  public function put($valeur) {
    $this->x = $valeur;
  }
}
?>
```

### d. Attribut ou méthode statique - Constante de classe

Un attribut ou une méthode statique sont utilisables directement sans instanciation préalable d'un objet. On parle aussi d'attribut ou de méthode de classe.

Pour définir un attribut ou une méthode statique, il suffit de placer le mot clé `static` devant la définition de l'attribut

ou de la méthode.

Pour référencer un attribut ou une méthode statique, il faut utiliser la syntaxe `nom_classe::$nom_attribut` ou `nom_classe::nom_méthode()`.

Une constante de classe est une constante définie dans une classe utilisable directement sans instanciation préalable d'un objet (comme un attribut de classe, mais constant).

*Syntaxe de définition d'une constante de classe*

```
const nom_constante = valeur;
```

`nom_constante`

Nom de la constante.

`valeur`

Valeur de la constante.

Une constante de classe est implicitement publique.

Pour référencer une constante de classe, il faut utiliser la syntaxe `nom_classe::nom_constante`.

*Exemple*

```
<?php
// Définition d'une classe.
class uneClasse {
  // Attribut privé quelconque.
  private $x;
  // Attribut privé statique pour stocker
  // le nombre d'objets instanciés.
  static private $nombre = 0;
  // Constante de classe pour définir une valeur
  // par défaut.
  const DEFAUT = 'X';
  // Fonction publique statique qui retourne le
  // nombre d'objets.
  static public function nombreObjets() {
    return uneClasse::$nombre;
  }
  // Méthode constructeur
  // - récupérer la valeur de l'attribut (valeur par défaut
  //   = la constante de classe)
  // - incrémenter le nombre d'objets
  public function __construct($valeur = uneClasse::DEFAUT) {
    $this->x = $valeur;
    uneClasse::$nombre++;
    echo "Création de l'objet : $this->x<br />";
  }
  // Méthode destructeur.
  // - décrémenter le nombre d'objets
  public function __destruct() {
    uneClasse::$nombre--;
    echo "Suppression de l'objet : $this->x<br />";
  }
}
// Créer deux objets.
$inconnu = new uneClasse();
$abc = new uneClasse ('ABC');
// Afficher le nombre d'objets
echo uneClasse::nombreObjets(),' objet(s)<br />';
// "Supprimer" un objet.
unset($inconnu);
// Afficher le nombre d'objets
echo uneClasse::nombreObjets(),' objet(s)<br />';
?>
```

*Résultat*

```
Création de l'objet : X
Création de l'objet : ABC
2 objet(s)
Suppression de l'objet : X
1 objet(s)
Suppression de l'objet : ABC
```

## 6. Exceptions

Les fonctionnalités orientées objet de PHP utilisent la notion d'exception pour gérer les erreurs (comme les langages C++ et Java).

Le principe consiste à inclure le code susceptible de générer des erreurs dans un bloc `try` et à lui associer un bloc `catch` destiné à intercepter les erreurs et à les traiter :

*Structure générale*

```
try {
  // Code susceptible de générer des erreurs.
  ...
}  catch (Exception $e) {
  // Code destiné à traiter les erreurs.
  ...
}
```

La classe `Exception` est une classe qui comporte notamment les méthodes suivantes :

`__construct`

Méthode constructeur acceptant deux paramètres : un message d'erreur et un code d'erreur (optionnel).

`getMessage`

Méthode permettant de récupérer le message d'erreur.

`getCode`

Méthode permettant de récupérer le code d'erreur.

À l'intérieur du bloc `try`, une exception peut être levée par une instruction du type `throw new Exception(message [,code])`. Dans le bloc `catch` les méthodes `getMessage` et `getCode` permettent de récupérer des informations sur l'erreur et de la traiter. En cas d'exception dans un bloc `try`, le traitement se branche directement dans le bloc `catch` : le reste du bloc `try` n'est pas exécuté.

*Exemple*

```
<?php
// Définition d'une classe.
class uneClasse {
  // Un attribut quelconque.
  private $x;
  // Méthode constructeur.
  public function __construct($valeur) {
    $this->x = $valeur;
  }
  // Méthode qui effectue une action quelconque.
  public function action() {
    // Pour une raison donnée l'action est interdite
    // si l'attribut est négatif : une exception est levée.
    if ($this->x < 0) {
      throw new Exception('Action interdite',123);
    }
  }
}
// Créer deux objets.
$objet = new uneClasse(1);
try {
```

```
  echo ’Objet 1 : ’;
  $objet->action(); // ne va pas lever d’exception
  echo ’OK<br />’;
} catch (Exception $e) {
  echo ’ERREUR ’,$e->getCode(),’ - ’,$e->getMessage(),’<br />’;
}
$objet = new uneClasse(-1);
try {
  echo ’Objet 2 : ’;
  $objet->action(); // va lever une exception
  echo ’OK<br />’;
} catch (Exception $e) {
  echo ’ERREUR ’,$e->getCode(),’ - ’,$e->getMessage(),’<br />’;
}
?>
```

*Résultat*

```
Objet 1 : OK
Objet 2 : ERREUR 123 - Action interdite
```

# Vue d'ensemble

Une erreur, dans un script PHP, peut se manifester de deux façons, éventuellement simultanées :

- par une valeur de retour particulière de la fonction PHP dans laquelle l'erreur est rencontrée ;
- par un message envoyé directement dans la page.

*Exemples*

| Fonction | Comportement en cas d'erreur |
|---|---|
| `require` | Si le fichier passé en paramètre n'existe pas, un message est affiché mais aucun code particulier n'est retourné par la fonction. |
| `mysqli_query` | Si le serveur MySQL retourne une erreur sur l'exécution d'une requête, aucun message n'est affiché mais la fonction retourne FALSE (la nature de l'erreur pouvant être récupérée par d'autres fonctions). |
| `mysqli_connect` | Si le serveur MySQL retourne une erreur lors de la connexion, un message est affiché et la fonction retourne FALSE (la nature de l'erreur pouvant être récupérée par d'autres fonctions). |

Gérer les erreurs dans un script PHP consiste donc, en général, à mettre en place un mécanisme qui permette de détecter la génération d'une erreur afin d'afficher soi-même un message à la place du message directement affiché par PHP.

# Les messages d'erreur PHP

Les messages d'erreur (ou d'alerte) affichés par PHP ont un niveau correspondant à leur gravité :

| Valeur | Constante associée | Description |
|---|---|---|
| 1 | `E_ERROR` | Erreur fatale d'exécution (message "fatal error: ..."). Le script ne s'exécute pas. Exemples : appel à une fonction qui n'existe pas, fichier mentionné dans l'instruction `require` qui n'existe pas. |
| 2 | `E_WARNING` | Alerte d'exécution (message "warning: ..."). Le script se poursuit. Exemple : tentative d'ouverture, avec `fopen`, d'un fichier qui n'existe pas, ouverture d'une connexion MySQL qui échoue... Généralement, la poursuite du script provoque d'autres messages du même type. |
| 4 | `E_PARSE` | Erreur de compilation ("Parse error: ..."). Le script ne s'exécute pas. Exemple : oubli d'un point-virgule, d'une parenthèse fermante... |
| 8 | `E_NOTICE` | Avertissement lors de l'exécution (message "Notice: ..."). Par défaut, PHP est configuré pour ne pas afficher ces avertissements. Le script se poursuit. Exemple : utilisation d'une variable non initialisée. |
| 16 | `E_CORE_ERROR` | Erreur fatale lors de l'initialisation de PHP. |
| 32 | `E_CORE_WARNING` | Alerte lors de l'initialisation de PHP. |
| 64 | `E_COMPILE_ERROR` | Erreur fatale lors de la compilation. |
| 128 | `E_COMPILE_WARNING` | Alerte lors de la compilation. |
| 256 | `E_USER_ERROR` | Erreur générée par le développeur. |
| 512 | `E_USER_WARNING` | Alerte générée par le développeur. |
| 1024 | `E_USER_NOTICE` | Avertissement généré par le développeur. |
| 2048 | `E_STRICT` | Conseils lors de l'exécution. Autorise PHP à suggérer des modifications pour améliorer la portabilité du code, notamment vers les futures versions (utilisation d'une fonction dépréciée par exemple). Semble masquer les erreurs de niveau inférieur. Ce niveau est apparu en version 5. |
| | | Erreur fatale récupérable. Si |

| | | |
|---|---|---|
| 4096 | E_RECOVERABLE_ERROR | l'erreur n'est pas gérée par le développeur (voir plus loin), le script s'interrompt. Ajouté en version 5.2.0. |
| 6143 | E_ALL | Toutes les erreurs et avertissements (somme des niveaux précédents), à l'exception de E_STRICT. En version 5, E_ALL valait 2047 : toutes les erreurs et avertissements, à l'exception de E_STRICT et E_RECOVERABLE_ERROR. |

*Exemple d'erreur fatale*

```
<?php
$fichier = fope('/tmp/info.txt','r');
$texte = fread($fichier,100);
fclose($fichier);
?>
```

*Résultat*

**Fatal error**: Call to undefined function: fope() in **/app/scripts/index.php** on line **2**

*Exemples d'alerte*

```
<?php
$fichier = fopen('/tmp/infos.txt','r');
$texte = fread($fichier,100);
fclose($fichier);
?>
```

*Résultat*

**Warning**: fopen(/tmp/infos.txt) [function.fopen]: failed to open stream: No such file or directory in **/app/scripts/index.php** on line **2**
**Warning**: fread(): supplied argument is not a valid stream resource in **/app/scripts/index.php** on line **3**
**Warning**: fclose(): supplied argument is not a valid stream resource in **/app/scripts/index.php** on line **4**

*Exemple d'erreur d'analyse*

```
<?php
echo 'Bonjour !<br />'     // pas de point virgule !
echo 'Bienvenue !<br />';
?>
```

*Résultat*

**Parse error**: syntax error, unexpected T_ECHO, expecting ',' or ';' in **/app/scripts/index.php** on line **3**

# Les fonctions de gestion des erreurs

PHP propose plusieurs fonctions permettant de gérer correctement les erreurs dans un script :

| Nom | Rôle |
|---|---|
| `error_reporting` | Définit les niveaux d'erreur qui sont affichés par PHP. |
| `error_log` | Envoie un message d'erreur vers une destination (fichier par exemple). |
| `set_error_handler` | Indique le nom d'une fonction utilisateur à utiliser comme gestionnaire d'erreurs. |
| `restore_error_handler` | Réactive l'ancien gestionnaire d'erreurs. |
| `trigger_error, user_error` | Déclenchent une erreur définie par le développeur (`user_error` est un alias de `trigger_error`). |
| `error_get_last` | Retourne des informations sur la dernière erreur rencontrée dans le script. |

En complément, l'opérateur `@`, placé devant le nom d'une fonction, permet de supprimer l'affichage des messages générés en cas d'erreur dans la fonction.

*Exemple*

```
<?php
$fichier = @fopen('/tmp/infos.txt','r');
$texte = @fread($fichier,100);
@fclose($fichier);
?>
```

Lors de l'exécution de ce script, aucun message n'est affiché bien que le fichier demandé n'existe pas.

Si l'erreur provoque l'arrêt du script, l'utilisation de l'opérateur `@` n'y change rien ; la page affichée est alors vide ou incomplète, et aucun message n'est affiché à l'utilisateur. Il convient donc de tester le résultat des fonctions, ou d'utiliser un gestionnaire d'erreurs, afin de contrôler le déroulement du programme et de faire les traitements nécessaires.

## error_reporting

La fonction `error_reporting` permet de définir les niveaux d'erreur pour lesquels le programme laisse PHP afficher les messages.

*Syntaxe*

```
entier error_reporting([entier niveaux])
```

`niveaux`

Niveaux d'erreur affichés par PHP, exprimés sous la forme d'une somme des valeurs affectées à chaque niveau.

La fonction `error_reporting` retourne l'ancienne valeur. Appelée sans paramètre, cette fonction se contente de retourner la valeur courante sans rien changer.

Pour définir les niveaux souhaités, il est vivement conseillé, pour la compatibilité future, d'utiliser les constantes et de ne pas mettre les valeurs en dur dans le programme.

La constante `E_ALL`, égale à la somme de toutes les autres constantes (sauf `E_STRICT`) peut être utilisée pour demander l'affichage de tous les niveaux d'erreur.

Inversement, un niveau égal à 0 provoque la suppression de l'affichage de tous les messages ; c'est l'équivalent, pour l'ensemble du script, de l'opérateur `@` qui peut être utilisé sur une fonction.

Telles que les valeurs de niveaux d'erreur sont définies, spécifier plusieurs niveaux se fait très simplement par des opérations arithmétiques sur les constantes.

*Exemple*

`E_ERROR+E_WARNING`

Niveaux `E_ERROR` et `E_WARNING`

`E_ALL-E_USER_ERROR-E_USER_WARNING-E_USER_NOTICE`

Tous les niveaux sauf E_USER_ERROR, E_USER_WARNING, et E_USER_NOTICE

La valeur par défaut (tout sauf E_NOTICE = E_ALL-E_NOTICE) est définie par la directive de configuration error_reporting.

En complément, la directive de configuration display_errors permet d'autoriser (on) ou d'interdire (off) l'affichage des messages d'erreur ; si display_errors est à off, donner une valeur quelconque à error_reporting (dans le fichier ini ou dans un script) est sans effet.

En phase de développement, il est conseillé d'afficher toutes les erreurs (E_ALL), y compris les avertissements, afin d'écrire un code le plus propre possible.

*Exemple*

```
<?php
// Valeur courante de error_reporting.
echo '<b>error_reporting = ',error_reporting(),'</b><br />';
// Par défaut égal à tout sauf E_NOTICE = E_ALL - E_NOTICE.
echo '= E_ALL - E_NOTICE = ',(E_ALL-E_NOTICE),'<br />';
// Affichage d'une variable non initialisée.
echo "\$x (non initialisée) = $x => pas de message <br />";
// Passage de error_reporting à E_ALL (tout)
error_reporting(E_ALL);
echo '<b>error_reporting = E_ALL</b><br />';
// Affichage d'une variable non initialisée.
echo "\$x (non initialisée) = $x => message <br />";
// Lecture d'un fichier qui n'existe pas.
if (! readfile('/tmp/infos.txt')) {
   echo 'Erreur dans readfile => message<br />';
};
// Passage de error_reporting à 0 (rien).
error_reporting(0);
echo '<b>error_reporting = 0</b><br />';
// Lecture d'un fichier qui n'existe pas.
if (! readfile('/tmp/infos.txt')) {
   echo 'Erreur dans readfile => plus de message<br />';
};
?>
```

*Résultat*

```
error_reporting = 6135
= E_ALL - E_NOTICE = 6135
$x (non initialisée) = => pas de message
error_reporting = E_ALL
Notice: Undefined variable: x in /app/scripts/index.php on line 12
$x (non initialisée) = => message
Warning: readfile(/tmp/infos.txt) [function.readfile]: failed
to open stream: No such file or directory in /app/scripts/index.php on line 14
Erreur dans readfile => message
error_reporting = 0
Erreur dans readfile => plus de message
```

En général, une fois en production, il est conseillé de désactiver l'affichage des messages d'erreur (soit par une directive de configuration, soit par un appel à error_reporting(0) au début de chaque script pour être indépendant de la configuration). Cette inhibition des messages d'erreurs, justifiée d'une part pour des raisons de sécurité (les messages d'erreurs PHP révèlent des informations sur l'arborescence du serveur) permet également d'afficher soi même des messages propres.

### error_log

La fonction error_log permet d'envoyer un message d'erreur vers une destination donnée.

*Syntaxe*

```
entier error_log(chaîne message, entier type_destination,
[chaîne destination[, chaîne complément])
```

message

Message à envoyer

type_destination

Type de destination : 0 : historique PHP ; 1 : adresse e-mail ; 2 : poste de déboggage ; 3 : fichier

destination

Précise la destination pour les types 1 à 3 : 1 : adresse e-mail ; 2 : adresse (IP ou nom) du poste distant et éventuellement numéro de port ; 3 : nom du fichier (créé automatiquement s'il n'existe pas)

`complément`

En-tête(s) complémentaire(s) à envoyer dans le message dans le cas 1 (voir la fonction `mail` dans le chapitre Utiliser les fonctions PHP - Envoyer un courrier électronique)

La fonction `error_log` retourne `TRUE` si le message a pu être envoyé et `FALSE` dans le cas contraire.

Pour le type 0 (historique PHP), le fichier de destination est défini par la directive de configuration `error_log`.

---

Si la directive de configuration `log_errors` est à `on`, les messages sont systématiquement écrits dans le fichier spécifié par la directive `error_log`, sans qu'il soit nécessaire d'appeler la fonction `error_log`.

---

Le type de destination 2 permet de mettre en place une « console » de déboggage sur un poste distant du serveur. Dans les grandes lignes, un petit programme doit être développé (en C, C++, VisualBasic ...) sur le poste en question, pour récupérer, sur un socket, les informations envoyées par le programme PHP. Cette possibilité ne sera pas étudiée plus en détail dans cet ouvrage.

*Exemple*

```
<?php
// Pas d'affichage des erreurs dans le script.
error_reporting(0);
// Lecture d'un fichier qui n'existe pas.
$nom_fichier = '/tmp/infos.txt';
if (! readfile($nom_fichier)) {
   // Ecriture d'un message d'erreur dans un fichier de trace
   // spécifique à l'application
   error_log("Impossible de lire le fichier $nom_fichier.\n",
             3,'/app/logs/monApplication.log');
   // Affichage d'un message pour l'utilisateur.
   echo 'Votre requête ne peut pas aboutir ; ',
        'essayez de nouveau plus tard.';
};
?>
```

*Résultat dans le fichier monApplication.log*

```
Impossible de lire le fichier /tmp/infos.txt.
```

*Résultat dans le navigateur*

```
Votre requête ne peut pas aboutir ; essayez de nouveau plus tard.
```

La fonction `error_log` est pratique, en phase de test ou de production, pour conserver la trace d'une erreur quelque part. Cependant, elle ne remplace pas l'affichage d'un message explicite pour l'utilisateur.

### set_error_handler

La fonction `set_error_handler` permet de spécifier le nom d'une fonction utilisateur qui doit être appelée pour gérer les erreurs de façon centralisée.

*Syntaxe*

```
chaîne set_error_handler(chaîne fonction[, entier niveaux])
```

*Avec*

`fonction`

Nom de la fonction chargée de gérer les erreurs

`niveaux`

Niveaux d'erreur concernés (apparu en version 5)

La fonction `set_error_handler` retourne le nom de l'ancienne fonction chargée de la gestion des erreurs (chaîne vide s'il n'y en avait pas) ou `FALSE` en cas d'erreur.

La fonction de gestion des erreurs doit accepter au minimum deux paramètres, le premier pour le niveau de l'erreur et le deuxième pour le message d'erreur. Trois paramètres supplémentaires peuvent être spécifiés pour le nom du fichier dans lequel l'erreur est apparue, le numéro de la ligne où l'erreur a été générée et le contexte de l'erreur (tableau de toutes les variables qui existaient au moment de l'erreur).

À partir du moment où un gestionnaire d'erreurs est spécifié, plus aucun message n'est affiché par PHP, quelle que soit la valeur de error_reporting. De plus, par défaut, le gestionnaire d'erreurs est appelé pour toutes les erreurs, quel que soit leur niveau et quelle que soit la valeur de error_reporting. Le paramètre niveaux permet de spécifier les niveaux concernés. Les niveaux E_ERROR, E_PARSE, E_CORE_ERROR, E_CORE_WARNING, E_COMPILE_ERROR et E_COMPILE_WARNING ne peuvent pas être gérés de cette façon.

*Exemple*

```
<?php
// Définir le gestionnaire d'erreurs.
function gestionnaire_erreurs
             ($niveau,$message,$fichier,$ligne) {
   // Afficher le fichier concerné, avec le numéro de ligne.
   echo "Fichier = $fichier<br />";
   echo "Ligne   = $ligne<br />";
   // Afficher le niveau et le message.
   echo "Niveau = $niveau <br />";
   echo "Message = $message<br />";
}
// Spécifier le gestionnaire à utiliser.
set_error_handler('gestionnaire_erreurs');
// Générer une erreur.
readfile('/tmp/infos.txt');
// Afficher un message de fin.
echo 'Fin';
?>
```

*Résultat*

```
Fichier = /app/scripts/index.php
Ligne = 15
Niveau = 2
Message = readfile(/tmp/infos.txt) [function.readfile]: failed to open stream: No such file or directory
Fin
```

Cet exemple montre que si l'erreur ne provoque pas l'interruption du script, ce dernier se poursuit après l'appel au gestionnaire d'erreurs ; il est donc de sa responsabilité, d'arrêter l'exécution du script, à l'aide de l'instruction exit ou de la fonction die (cf. chapitre Introduction à PHP - Les bases du langage PHP).

*Exemple*

```
<?php
// Définir le gestionnaire d'erreurs.
function gestionnaire_erreurs
             ($niveau,$message,$fichier,$ligne) {
   // Afficher le fichier concerné, avec le numéro de ligne.
   echo "Fichier = $fichier<br />";
   echo "Ligne   = $ligne<br />";
   // Afficher le niveau et le message.
   echo "Niveau = $niveau <br />";
   echo "Message = $message<br />";
   // Interrompre le script
   exit;
}
// Spécifier le gestionnaire à utiliser.
set_error_handler('gestionnaire_erreurs');
// Générer une erreur.
readfile('/tmp/infos.txt');
// Afficher un message de fin.
echo 'Fin';
?>
```

*Résultat*

```
Fichier = /app/scripts/index.php
Ligne = 15
Niveau = 2
Message = readfile(/tmp/infos.txt) [function.readfile]: failed to open stream: No such file or directory
```

Dans le gestionnaire d'erreurs, il est possible d'utiliser la fonction header (cf. chapitre Gérer les liens et les formulaires avec PHP - Aller sur une autre page) pour rediriger l'utilisateur vers une page HTML ou un script PHP chargé de l'affichage des messages d'erreurs.

Cette technique est très pratique car elle permet de centraliser la gestion des erreurs (la fonction peut être définie dans un fichier inclus) et de séparer clairement le code chargé du traitement normal, du code chargé de gérer les erreurs.

## restore_error_handler

La fonction restore_error_handler permet de restaurer l'ancien gestionnaire d'erreurs après un changement effectué avec set_error_handler.

*Syntaxe*

```
restore_error_handler()
```

*Exemple*

```
<?php
// Définir un premier gestionnaire d'erreurs.
function gestionnaire1 ($numéro,$message) {
   // Affiche un simple message
   echo '=> gestionnaire n° 1<br />';
}
// Définir un deuxième gestionnaire d'erreurs.
function gestionnaire2 ($numéro,$message) {
   // Affiche un simple message
   echo '=> gestionnaire n° 2<br />';
}
// Définir une fonction qui génère une erreur.
function générer_erreur() {
   // Afficher un message.
   echo 'Générer une erreur<br />';
   // Lire un fichier qui n'existe pas.
   readfile('/tmp/infos.txt');
}
// Première séquence : pas de gestionnaire.
echo '<b>Pas de gestionnaire</b><br />';
générer_erreur();
// Deuxième séquence : gestionnaire numéro 1.
set_error_handler('gestionnaire1');
echo '<b>Utiliser le gestionnaire n° 1</b><br />';
générer_erreur();
// Troisième séquence : gestionnaire numéro 2.
set_error_handler('gestionnaire2');
echo '<b> Utiliser le gestionnaire n° 2</b><br />';
générer_erreur();
// Quatrième séquence : restaurer l'ancien gestionnaire.
restore_error_handler();
echo '<b>Premier restore_error_handler()</b><br />';
générer_erreur();
// Cinquième séquence : restaurer l'ancien gestionnaire.
restore_error_handler();
echo '<b>Deuxième restore_error_handler()</b><br />';
générer_erreur();
?>
```

*Résultat*

```
Pas de gestionnaire
Générer une erreur
Warning: readfile(/tmp/infos.txt) [function.readfile]: failed
to open stream: No such file or directory in /app/scripts/index.php on
line 17
Utiliser le gestionnaire n° 1
Générer une erreur
=> gestionnaire n° 1
Utiliser le gestionnaire n° 2
Générer une erreur
=> gestionnaire n° 2
Premier restore_error_handler()
Générer une erreur
=> gestionnaire n° 1
Deuxième restore_error_handler()
Générer une erreur
Warning: readfile(/tmp/infos.txt) [function.readfile]: failed
to open stream: No such file or directory in /app/scripts/index.php
on line 17
```

Cet exemple illustre la possibilité d'empiler et de désempiler les gestionnaires d'erreurs grâce aux fonctions set_error_handler et restore_error_handler.

Si une partie d'un script a besoin d'un gestionnaire différent de celui utilisé dans le reste du script, il suffit d'appeler `set_error_handler` au début de la section concernée pour définir le gestionnaire puis `restore_error_handler` à la fin de la section pour remettre l'ancien.

Par ailleurs, il faut noter que le gestionnaire par défaut, c'est-à-dire celui de PHP, est systématiquement replacé à la fin de l'exécution d'un script ; il n'est donc pas nécessaire d'appeler `restore_error_handler` en fin de script.

### trigger_error (ou son alias user_error)

La fonction `trigger_error` permet de déclencher une erreur définie par le développeur comme si cette erreur avait été déclenchée nativement par PHP.

*Syntaxe*

```
trigger_error(chaîne message, entier niveau)
```

`message`

Message de l'erreur.

`niveau`

Niveau de l'erreur parmi `E_USER_EROR`, `E_USER_WARNING` et `E_USER_NOTICE` (toute autre valeur génère une erreur !)

L'erreur déclenchée par `trigger_error` est traitée par le gestionnaire interne de PHP (affichage du message) ou le gestionnaire éventuellement défini par `set_error_handler`. Dans le premier cas, le script est interrompu si le niveau de l'erreur est égal à `E_USER_ERROR` (sinon il se poursuit). Dans le deuxième cas, le script se poursuit quel que soit le niveau de l'erreur (c'est au gestionnaire d'interrompre le script si besoin).

*Exemple avec le gestionnaire interne*

```
<?php
// Déclencher une erreur E_USER_NOTICE.
trigger_error('*** mon message ***',E_USER_NOTICE);
// Déclencher une erreur E_USER_WARNING.
trigger_error('*** mon message ***',E_USER_WARNING);
// Déclencher une erreur E_USER_ERROR.
trigger_error('*** mon message ***',E_USER_ERROR);
// Afficher un message de fin.
echo 'Fin';

?>
```

*Résultat*

```
Notice: *** mon message *** in /app/scripts/index.php on line 3
Warning: *** mon message *** in /app/scripts/index.php on line 5
Fatal error: *** mon message *** in /app/scripts/index.php on line 7
```

Cet exemple montre que les deux premières erreurs (niveaux `E_USER_NOTICE` et `E_USER_WARNING`) ne provoquent pas l'arrêt du script, à la différence de la troisième (niveau `E_USER_ERROR`).

*Exemple avec un gestionnaire externe*

```
<?php
// Définir le gestionnaire d'erreurs.
function gestionnaire_erreurs($niveau,$message) {
   // Afficher simplement le niveau et le message.
   echo "Niveau = $niveau <br />";
   echo "Message = $message<br />";
   // Ne pas interrompre le script.
   // exit;
}
// Spécifier le gestionnaire à utiliser.
set_error_handler('gestionnaire_erreurs');
// déclencher une erreur E_USER_NOTICE
trigger_error('*** mon message ***',E_USER_NOTICE);
// déclencher une erreur E_USER_WARNING
trigger_error('*** mon message ***',E_USER_WARNING);
// déclencher une erreur E_USER_ERROR
trigger_error('*** mon message ***',E_USER_ERROR);
// afficher un message de fin
echo 'Fin';

?>
```

*Résultat*

```
Niveau = 1024
Message = *** mon message ***
Niveau = 512
Message = *** mon message ***
Niveau = 256
Message = *** mon message ***
Fin
```

## error_get_last

La fonction `error_get_last` retourne des informations sur la dernière erreur rencontrée dans le script. Cette fonction est apparue en version 5.2.3.

*Syntaxe*

```
tableau error_get_last()
```

La fonction `error_get_last` retourne un tableau associatif contenant les clés `type`, `message`, `file` et `line`.

*Exemple*

```
<?php
// Générer une erreur (sans l'afficher : @...).
@readfile('/tmp/infos.txt');
// Poursuivre le script.
echo 'suite ...<br />';
// Afficher des informations sur la dernière erreur.
foreach (error_get_last() as $clé => $valeur) {
  echo "$clé => $valeur<br />";
}
?>
```

*Résultat*

```
suite ...
type => 2
message => readfile(/tmp/infos.txt) [function.readfile]: failed to open
stream: No such file or directory
file => /app/scripts/index.php
line => 3
```

## La variable $php_errormsg

Si la directive de configuration `track_errors` est à `on`, le message de la dernière erreur est disponible dans la variable `$php_errormsg`.

L'utilisation de cette variable directement dans un script est un substitut possible à la mise en place d'un gestionnaire d'erreurs.



Il existe de nombreuses directives de configuration relatives aux erreurs, en plus de celles présentées dans ce chapitre. Pour plus d'informations, reportez-vous à la documentation de PHP.

# Vue d'ensemble

## 1. Introduction

Dans les sites Web dynamiques et *interactifs*, il est très souvent nécessaire d'interagir avec l'utilisateur.

En HTML, il existe principalement deux méthodes pour interagir avec un utilisateur :

- les liens (balise `<a>`) ;
- les formulaires (balise `<form>`).

Des scripts PHP peuvent être utilisés pour traiter le clic de l'utilisateur sur un lien ou la saisie de l'utilisateur dans un formulaire.

## 2. Les liens

Le lien est la technique de base qui permet à un utilisateur de naviguer entre les différentes pages d'un site.

Un lien HTML est défini entre les balises `<a>` et `</a>`.

*Syntaxe simplifiée*

```
<a
  [ action="url" ]
  [ id="identifiant_lien" ]
  [ target="cible" ]
>
...
</a>
```

Les attributs de la balise `<a>` sont les suivants :

`action`

URL (*Uniform Resource Locator*) relative ou absolue qui est appelée par le lien, en ce qui nous concerne, un script PHP.

`id`

Identifiant du lien. Si la page HTML contient plusieurs liens, l'identifiant permet de les différencier. En ce qui nous concerne, cet identifiant ne présente pas d'intérêt car il n'est pas récupéré dans le script de traitement du lien. Par contre, il peut être utilisé côté client, en JavaScript par exemple.

`target`

Cible (par exemple une autre fenêtre) dans laquelle ouvrir l'URL cible. Par défaut, l'URL cible s'affiche dans la même fenêtre.

L'URL peut contenir des paramètres qui permettent de passer des informations d'une page à une autre.

*Syntaxe*

```
url_classique?nom=valeur[&...]
```

Le point d'interrogation (`?`) introduit la liste des paramètres de l'URL séparés par le caractère éperluette (`&`) ; chaque paramètre est constitué par un couple nom/valeur sous la forme `nom=valeur` :

```
www.monsite.com/info/accueil.php?prenom=Olivier
chercher.php?prenom=Olivier&nom=HEURTEL
```

*Exemple*

- Script page1.php

```
<?php
// Initialisation d'une variable.
$nom='Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=<?php echo $nom; ?>">Page 2</a>
    </div>
  </body>
</html>
```

- Source de la page dans le navigateur

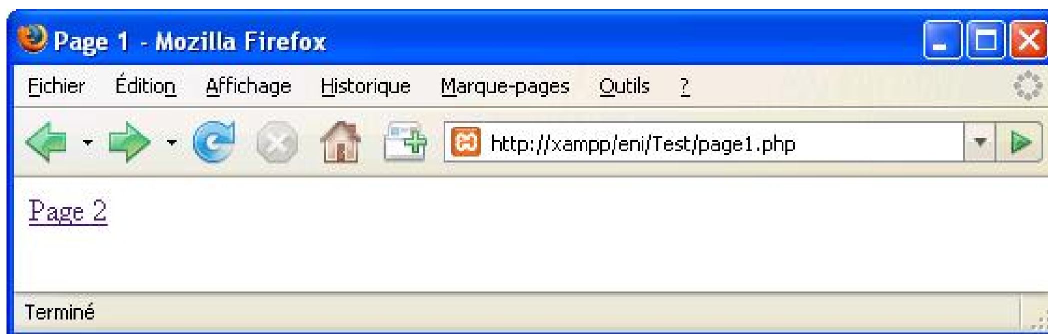
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=Olivier">Page 2</a>
    </div>
  </body>
</html>
```

- Script page2.php

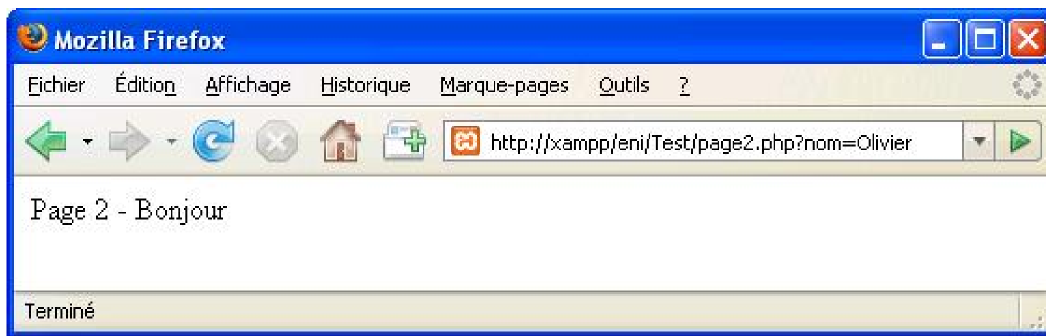
```
<?php
echo "Page 2 - Bonjour $nom";
?>
```

*Résultat*

- Affichage de la page 1




- Résultat du clic sur le lien




Pour l'instant, aucun nom n'est affiché dans la deuxième page. La variable `$nom` définie dans le script `page1.php` n'est pas disponible dans le script `page2.php` (cf. chapitre Introduction à PHP - Les bases du langage PHP sur la portée et la durée de vie des variables). De plus, notre script ne contient aucune instruction permettant de récupérer les données passées dans l'URL ; nous verrons comment procéder à la section Récupérer les données d'une URL ou d'un formulaire.

## 3. Les formulaires

### a. Petit rappel sur les formulaires

Le formulaire est un outil de base indispensable pour les sites Web dynamiques puisqu'il permet à l'utilisateur de saisir des informations et donc d'interagir avec le site.

Un formulaire HTML est défini entre les balises `<form>` et `</form>`.

*Syntaxe simplifiée*

```
<form
   [ action="url_de_traitement" ]
   [ method="GET"|"POST" ]
   [ id="identifiant_formulaire" ]
   [ target="cible" ]
>
...
</form>
```

Les attributs de la balise `<form>` sont les suivants :

`action`

URL relative ou absolue (*Uniform Resource Locator*) qui va traiter le formulaire, en ce qui nous concerne, un script PHP. Cet attribut est obligatoire pour se conformer à la recommandation XHTML stricte.

`method`

Mode de transmission vers le serveur des informations saisies dans le formulaire. `GET` (valeur par défaut) : les données du formulaire sont transmises dans l'URL. `POST` : les données du formulaire sont transmises dans le corps de la requête.

`id`

Identifiant du formulaire. Si la page HTML contient plusieurs formulaires, l'identifiant permet de les différencier. En ce qui nous concerne, cet identifiant ne présente pas d'intérêt car il n'est pas récupéré dans le script de traitement du formulaire. Par contre, il peut être utilisé côté client, en JavaScript par exemple.

`target`

Cible (par exemple une autre fenêtre) dans laquelle ouvrir l'URL cible.

Entre les balises `<form>` et `</form>`, il est possible de placer des balises `<input>`, `<select>` ou `<textarea>` pour définir des zones de saisies.

*Exemple (formulaire HTML complet)*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Saisie</title>
  </head>
  <body>
    <form action=" " method="post">
    <div>
      Nom :
      <input type="text" name="nom" value=""
           size="20" maxlength="20" />
      Mot de passe :
      <input type="password" name="mot_de_passe" value=""
           size="20" maxlength="20" />
      <br />Sexe :
      <input type="radio" name="sexe" value="M" />Masculin
      <input type="radio" name="sexe" value="F" />Feminin
      <input type="radio" name="sexe" value="?"
                        checked="checked" />Ne sait pas
      <br />Photo :
      <input type="file" name="photo" value="" size="50" />
      <br />Couleurs préférées :
      <input type="checkbox" name="couleurs[bleu]" />Bleu
      <input type="checkbox" name="couleurs[blanc]" />Blanc
      <input type="checkbox" name="couleurs[rouge]" />Rouge
      <input type="checkbox" name="couleurs[pas]"
                            checked="checked" />Ne sait pas
      <br />Langue :
      <select name="langue">
        <option value="E">Espagnol</option>
        <option value="F" selected="selected" >Francais</option>
        <option value="I">Italien</option>
      </select>
      <br />Fruits préféres :<br />
      <select name="fruits[]" multiple="multiple" size="8">
        <option value="A">Abricots</option>
        <option value="C">Cerises</option>
        <option value="F">Fraises</option>
        <option value="P">Pêches</option>
        <option value="?" selected="selected">
                          Ne sait pas</option>
      </select>
      <br />Commentaire :<br />
      <textarea name="commentaire" rows="4" cols="50"></textarea>
      <br />
      <input type="hidden" name="invisible" value="123" /><br />
      <input type="submit" name="soumettre" value="OK" />
      <input type="image" name="valider" src="valider.gif" />
      <input type="reset" name="effacer" value="Effacer" />
      <input type="button" name="action" value="Ne fait rien" />
    </div>
    </form>
  </body>
</html>
```

*Résultat*

Nom : Mot de passe :
Sexe : Masculin Feminin Ne sait pas
Photo : Parcourir...
Couleurs préférées : Bleu Blanc Rouge Ne sait pas
Langue : Francais
Fruits préféres :
Abricots
Cerises
Fraises
Pêches
Ne sait pas
Commentaire :

OK Effacer Ne fait rien

PHP peut intervenir à deux endroits par rapport au formulaire :

- pour la construction du formulaire, si ce dernier doit contenir des informations dynamiques ;
- pour le traitement du formulaire (c'est-à-dire des données saisies par l'utilisateur dans le formulaire).

## b. Construire un formulaire dynamiquement

Comme tout le reste de la page, tout ou partie d'un formulaire peut être construit dynamiquement. Trois cas seront abordés dans ce paragraphe :

- générer la totalité du formulaire ;
- générer des valeurs initiales dans les zones de saisie ;
- générer une liste d'options.

### Générer la totalité du formulaire

S'il existe une description du formulaire sous une forme ou sous une autre, il est possible de générer la totalité du formulaire.

Dans l'exemple simplifié suivant, nous supposons que nous récupérons (dans un fichier, dans une base...) une description du formulaire sous la forme d'un tableau à deux dimensions : chaque ligne du tableau contient une description de la zone sous la forme d'un tableau avec le titre, le type, le nom et la valeur.

```
<?php
// Tableau contenant la description du formulaire.
$formulaire = array(
  array('Nom : ','text','nom','HEURTEL'),
  array('','submit','ok','OK') );
// Génération du formulaire à l'aide d'une boucle
// sur le tableau.
echo '<form action="saisie.php" method="POST">';
foreach($formulaire as $zone) {
```

```
  echo "$zone[0]<input type=\"$zone[1]\" ",
       "name=\"$zone[2]\" value=\"$zone[3]\"><br />";
}
echo '</form>';
?>
```

*Résultat à l'écran*

Nom : HEURTEL
OK

*Résultat dans le source du navigateur*

```
<form action="saisie.php" method="POST">Nom : <input
type="text" name="nom" value="HEURTEL"><br /><input
type="submit" name="ok" value="OK"><br /></form>
```

## Générer des valeurs initiales dans les zones de saisie

Nous avons déjà étudié cette possibilité dans différents exemples.

*Exemple*

```
<form action="saisie.php" method="POST">
Nom : <input type="text" name="nom"
            value="<?php echo $nom ?>"><br />
<input type="submit" name="ok" value="OK">
</form>
```

Ici, nous supposons que $nom est une variable initialisée par ailleurs dans le script PHP.

## Générer une liste d'options

Du code PHP peut être utilisé pour générer des listes d'options, soit dans une zone <select> (liste à sélection unique ou multiple), soit une zone <input> de type radio (groupe de boutons radio) soit dans une zone <input> de type checkbox (cases à cocher).

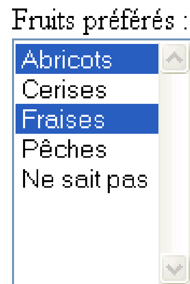
Les données affichées proviennent très souvent d'une base de données et il est intéressant de pouvoir construire cette zone du formulaire dynamiquement à partir des données extraites de la base.

*Exemple avec une liste à sélection multiple*

```
<?php
// Liste des fruits à afficher dans la liste, sous la
// forme d'un tableau associatif donnant le code du
// fruit (clé du tableau) et l'intitulé du fruit.
$fruits_du_marché = array(
  'A' => 'Abricots',
  'C' => 'Cerises',
  'F' => 'Fraises',
  'P' => 'Pêches',
  '?' => 'Ne sait pas');
// Liste des fruits préférés de l'utilisateur, sous la
// forme d'un tableau donnant le code des fruits concernés.
$fruits_préférés = array('A','F');
// Remarque : nous verrons ultérieurement comment récupérer
//            ces informations dans une base.
?>
<!-- construction du formulaire -->
<form action="saisie.php" method="POST">
Fruits préférés :<br />
<select name="fruits[]" multiple size="8">
<?php
// Code PHP générant la partie dynamique du formulaire.
// Parcourir la liste à afficher et récupérer le code
// et l'intitulé.
foreach($fruits_du_marché as $code => $intitulé) {
```

```
  // Déterminer si la ligne doit être sélectionnée
  //  - oui si le code figure dans la liste des fruits
  //    préférés de l'utilisateur => recherche de $code
  //    dans $fruits_préférés avec la fonction in_array
  //  - si c'est le cas, mettre l'attribut "selected" dans
  //    la balise "option", sinon ne rien mettre.
  $sélection =
    in_array($code,$fruits_préférés)?'selected="selected"':'';
  // Générer la balise "option" avec la variable $code pour
  // l'attribut "value", la variable $sélection pour
  // l'indication de sélection et la variable $intitulé
  // pour le texte affiché dans la liste.
  echo "<option value=\"$code\" $sélection>$intitulé</option>";
}
?>
</select>
</form>
```

*Résultat à l'écran*




*Source dans le navigateur*

```
<!-- construction du formulaire -->
<form action="saisie.php" method="POST">
Fruits préférés :<br />
<select name="fruits[]" multiple size="8">
<option value="A" selected="selected">Abricots</option><option value="C"
>Cerises</option><option value="F"
selected="selected">Fraises</option><option value="P"
>Pêches</option><option value="?" >Ne sait pas</option></select>
</form>
```

Cet exemple est très facile à adapter si l'attribut `value` n'est pas utilisé.

*Exemple avec une liste à sélection unique*

```
<?php
// Liste des langues à afficher dans la liste, sous la
// forme d'un tableau associatif donnant le code de
// la langue (clé du tableau) et l'intitulé de la langue.
$langues_disponibles = array(
  'E' => 'Espagnol',
  'F' => 'Français',
  'I' => 'Italien');
// Code de la langue de l'utilisateur
$langue = 'F';
?>
<!-- construction du formulaire -->
<form action="saisie.php" method="POST">
Langue :<br />
<select name="langue">
<?php
// Code PHP générant la partie dynamique du formulaire.
// Parcourir la liste à afficher et récupérer le code
// et l'intitulé.
foreach($langues_disponibles as $code => $intitulé) {
```

```
  // Déterminer si la ligne doit être sélectionnée
  //  - oui si le code est égal au code de la langue de
  //    l'utilisateur
  //  - si c'est le cas, mettre l'attribut "selected" dans
  //    la balise "option", sinon ne rien mettre
  $sélection = ($code == $langue)?'selected="selected"':'';
  // Générer la balise "option" avec la variable $code pour
  // l'option "value", la variable $sélection pour
  // l'indication de sélection et la variable $intitulé
  // pour le texte affiché dans la liste.
  echo "<option value=\"$code\" $sélection>$intitulé</option>";
}
?>
</select>
</form>
```

*Résultat à l'écran*

*Source dans le navigateur*

```
<!-- construction du formulaire -->
<form action="saisie.php" method="POST">
Langue :<br />
<select name="langue">
<option value="E" >Espagnol</option><option value="F"
selected="selected">Français</option><option value="I"
>Italien</option></select>
</form>
```

Des techniques similaires peuvent être utilisées pour construire une liste de cases à cocher, un groupe de boutons radio, etc.

### c. Traiter un formulaire à l'aide d'un script PHP

Il existe principalement trois méthodes pour traiter un formulaire à l'aide d'un script PHP :

- placer le formulaire dans un document HTML "pur" (`.htm` ou `.html`) et indiquer le nom du script qui doit traiter le formulaire dans l'attribut `action` de la balise `<form>` : dans ce cas le formulaire ne contient aucun élément dynamique ;
- placer le formulaire dans un script PHP (par exemple, pour construire une partie du formulaire dynamiquement) et faire traiter le formulaire par un autre script PHP (mentionné dans l'attribut `action` de la balise `<form>`) ;
- Placer le formulaire dans un script PHP (par exemple, pour construire une partie du formulaire dynamiquement), et le faire traiter par le même script PHP (mentionné dans l'option `action` de la balise `<form>`).

Par ailleurs, quelque part sur une autre page, un lien (Saisie par exemple) peut être inséré pour appeler le formulaire de saisie :

- Formulaire HTML :

```
<a href="saisie.htm">Saisie</a>
```

- Formulaire PHP :

```
<a href="saisie.php">Saisie</a>
```

## Première méthode

*Document HTML `saisie.htm`*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="traitement.php" method="post">
    <div>
      Nom : <input type="text" name="nom" value="" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Script PHP `traitement.php`*

```
<?php
/* A faire ...
    - récupérer les informations saisies
    - faire le traitement
    - afficher une nouvelle page
*/
?>
```

*Résultat*

- Affichage initial du formulaire :

- Saisie d'une information :

- Le résultat du clic sur le bouton **OK** est une page vide, car, pour l'instant, le script de traitement ne fait rien.

## Deuxième méthode

*Document PHP `saisie.php`*

Un peu de code PHP (en gras) est utilisé pour générer une partie dynamique du formulaire.

```
<?php
// Inclure un fichier qui contient des définitions de
// constantes, dont le titre de la page (TITRE_PAGE_SAISIE).
require('constantes.inc');
// Initialisation d'une variable qui contient la valeur
// initiale de la zone de saisie (dans la pratique, cette
// valeur vient sans doute d'ailleurs et n'est pas codée
// en dur).
$nom = 'X';
// Dans le code HTML qui suit, inclusion de deux petits
// bouts de code PHP pour afficher respectivement le titre
// de la page et la valeur initiale de la zone de saisie.
?>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title><?php echo TITRE_PAGE_SAISIE; ?></title>
  </head>
  <body>
    <form action="traitement.php" method="post">
    <div>
      Nom : <input type="text" name="nom"
                      value="<?php echo $nom; ?>" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Script PHP `traitement.php`*

```
<?php
/* A faire ...
    - récupérer les informations saisies
    - faire le traitement
    - afficher une nouvelle page
*/
?>
```

*Résultat*

- Affichage initial du formulaire (une valeur initiale dynamique est proposée pour la zone de saisie) :

Nom : X OK

- Saisie d'une information :

Nom : Olivier OK

- Le résultat du clic sur le bouton **OK** est une page vide, car, pour l'instant, le script de traitement ne fait rien.

## Troisième méthode

*Document PHP `saisie.php`*

C'est le même script que précédemment, en ayant simplement changé l'attribut `action` de la balise `<form>` pour indiquer que le formulaire doit être traité par le même script `saisie.php`.

```
<?php
// Inclure un fichier qui contient des définitions de
// constantes, dont le titre de la page (TITRE_PAGE_SAISIE).
require('constantes.inc');
// Initialisation d'une variable qui contient la valeur
// initiale de la zone de saisie (dans la pratique, cette
// valeur vient sans doute d'ailleurs et n'est pas codée
// en dur).
$nom = 'X';
// Dans le code HTML qui suit, inclusion de deux petits
// bouts de code PHP pour afficher respectivement le titre
// de la page et la valeur initiale de la zone de saisie.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
    <title><?php echo TITRE_PAGE_SAISIE; ?></title>
  </head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom : <input type="text" name="nom"
                      value="<?php echo $nom; ?>" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Résultat*

- Affichage initial du formulaire (une valeur initiale dynamique est proposée pour la zone de saisie) :

Nom : X OK

- Saisie d'une information :

Nom : Olivier OK

- Le résultat du clic sur le bouton **OK** est la même page de nouveau affichée, car, pour l'instant, le script de traitement ne fait rien de plus :

Nom : X OK

**Que choisir ?**

Le choix de telle ou telle méthode dépend de la complexité du site et des préférences de chacun.

Quelques considérations générales :

- Séparer la page HTML (ou le script PHP qui génère le formulaire) du script PHP qui le traite a un inconvénient au niveau de la maintenance : si des modifications sont apportées au formulaire, il y a deux fichiers à modifier (avec des risques d'erreur, d'oubli).

- Inversement, si le formulaire n'a aucune partie dynamique, l'écrire dans un fichier HTML séparé du script PHP qui le traite permet de bien séparer l'interface utilisateur (la couche "présentation") du traitement.

- Dans la pratique, pour faciliter la maintenance, il est souhaitable de définir certaines valeurs affichées à plusieurs reprises (nom de société par exemple) dans des constantes ou des variables et d'utiliser ces constantes/variables dans les pages : toutes les pages deviennent un peu dynamiques et la troisième méthode nous parait optimale.

Dans la suite de ce chapitre, nous allons rentrer dans le détail du traitement du formulaire en PHP, en utilisant des exemples construits sur le modèle de la troisième méthode.

## 4. Récupérer les données d'une URL ou d'un formulaire

À la différence des scripts CGI, il n'y pas besoin de faire des analyses complexes de chaînes de caractères ("parser") pour récupérer les valeurs passées dans une URL ou saisies par l'utilisateur dans un formulaire ; ces valeurs vont être récupérées très simplement dans le script de traitement.

### a. Première méthode : les tableaux $_POST, $_GET et $_REQUEST

Par défaut, toutes les zones de formulaire sont automatiquement enregistrées dans le script PHP qui traite le

formulaire, dans un tableau associatif `$_POST` pour les formulaires `POST` et `$_GET` pour les formulaires `GET` : la clé du tableau est égale au nom de la zone dans le formulaire (option `name` de la balise `<input>`, `<select>` ou `<textarea>`) et la valeur égale à la valeur saisie dans la zone.

De même, tous les paramètres de l'URL sont enregistrés dans le tableau associatif `$_GET` : la clé du tableau est égale au nom du paramètre et la valeur égale à la valeur passée dans l'URL.

En complément, ces informations sont aussi disponibles dans le tableau associatif `$_REQUEST` qui regroupe le contenu des tableaux `$_GET` et `$_POST` (et nous le verrons plus tard du tableau `$_COOKIE` qui contient des informations sur les cookies).

---

Les tableaux `$_GET`, `$_POST` et `$_REQUEST` sont apparus en version 4.1. Avant la version 4.1, `$_GET` et `$_POST` s'appelaient `$HTTP_GET_VARS` et `$HTTP_POST_VARS`, et `$_REQUEST` n'avait pas d'équivalent. Pour des raisons de compatibilité ascendante, ces variables "longues" sont toujours disponibles à condition de positionner la directive de configuration `register_long_array` à `on` (`off` par défaut).

---

### Exemple avec un formulaire

*Document HTML saisie.htm*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="traitement.php" method="post">
    <div>
      Nom : <input type="text" name="nom" value="" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Script PHP traitement.php*

```
<?php
// Affichage des informations contenues dans les
// tableaux $_POST et $_REQUEST.
echo '$_POST[\'nom\'] -> ',$_POST['nom'],'<br \>';
echo '$_REQUEST[\'nom\'] -> ',$_REQUEST['nom'],'<br \>';
?>
```

*Résultat*

- Affichage initial du formulaire

Nom : [ ] OK

- Saisie d'une valeur

Nom : Olivier OK

- Résultat du clic sur le bouton **OK**

```
$_POST['nom'] -> Olivier
$_REQUEST['nom'] -> Olivier
```

### Exemple avec une URL

*Script page1.php*

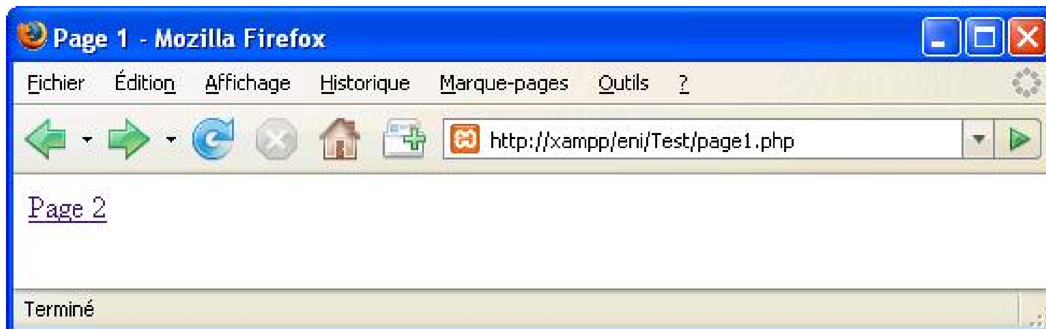
```
<?php
// Initialisation d'une variable.
$nom='Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=<?php echo $nom; ?>">Page 2</a>
    </div>
  </body>
</html>
```

*Script page2.php*

```
<?php
// Affichage des informations contenues dans les
// tableaux $_GET et $_REQUEST.
echo '$_GET[\'nom\'] -> ',$_GET['nom'],'<br \>';
echo '$_REQUEST[\'nom\'] -> ',$_REQUEST['nom'],'<br \>';
?>
```

*Résultat*

- Affichage de la page 1




- Résultat du clic sur le lien

```
$_GET['nom'] -> Olivier
$_REQUEST['nom'] -> Olivier
```

### b. Deuxième méthode : la fonction import_request_variables

Les informations passées dans une URL ou saisies dans un formulaire peuvent être automatiquement importées dans des variables PHP grâce à la fonction `import_request_variables`.

*Syntaxe*

```
import_request_variables(chaîne types,chaîne préfixe)
```

`types`

Type d'information désirée : `G` ou `g` : informations envoyées par la méthode `GET` ; `P` ou `p` : informations envoyées par la méthode `POST` ; `C` ou `c` : informations envoyées par un cookie (cf. chapitre Gérer les sessions - Utiliser des cookies).

`préfixe`

Préfixe à ajouter devant le nom de la variable.

Lorsque cette fonction est appelée dans le script de traitement d'une URL, PHP crée une variable pour chaque paramètre de l'URL : le nom de la variable est égal au nom du paramètre dans l'URL précédé du préfixe, et la valeur de la variable est égale à la valeur passée dans l'URL.

Lorsque cette fonction est appelée dans le script de traitement d'un formulaire, PHP crée une variable pour chaque zone du formulaire : le nom de la variable est égal au nom de la zone dans le formulaire précédé du préfixe, et la valeur de la variable est égale à la valeur saisie dans le formulaire.

Le préfixe est optionnel mais une erreur de niveau `E_NOTICE` est générée s'il n'est pas renseigné.

Vous pouvez utiliser une combinaison des différentes lettres de types pour importer en une seule fois des informations d'origines différentes. Par exemple, spécifier `PG` permet de récupérer les informations envoyées par une méthode `POST` ou par une méthode `GET`.

### Exemple avec un formulaire

*Document HTML saisie.htm*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="traitement.php" method="post">
    <div>
      Nom : <input type="text" name="nom" value="" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Script PHP traitement.php*

```
<?php
// Importer les informations du formulaire :
//   - méthode POST
//   - préfixe form_
import_request_variables('P','form_');
// Afficher la variable importée.
echo '$form_nom = ',$form_nom,'<br />';
?>
```

*Résultat*

- Affichage initial du formulaire

Nom : [ ] OK

- Saisie d'une valeur

Nom : Olivier OK

- Résultat du clic sur le bouton **OK**

```
$form_nom = Olivier
```

### Exemple avec une URL

*Script page1.php*

```
<?php
// Initialisation d'une variable.
```

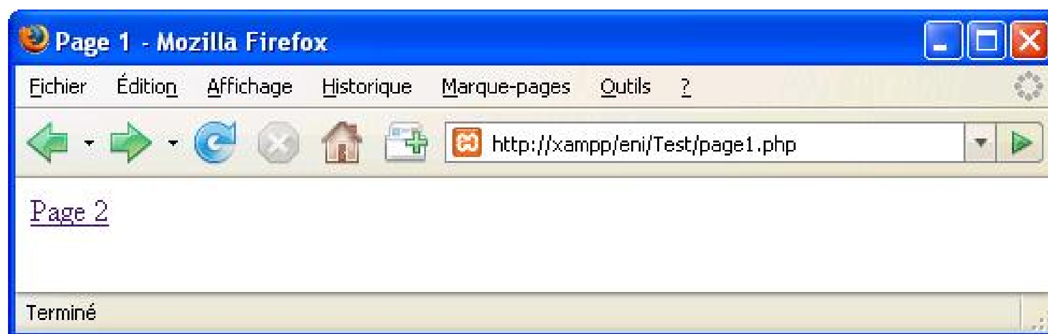
```
$nom='Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=<?php echo $nom; ?>">Page 2</a>
    </div>
  </body>
</html>
```

*Script page2.php*

```
<?php
// Importer les informations de l'URL :
//   - méthode GET
//   - préfixe url_
import_request_variables('G','url_');
// Afficher la variable importée.
echo '$url_nom = ',$url_nom,'<br />';
?>
```

*Résultat*

- Affichage de la page 1




- Résultat du clic sur le lien

```
$url_nom = Olivier
```

## c. Troisième méthode (non recommandée)

La troisième méthode est non recommandée car elle est dépendante d'une directive de configuration et elle peut poser des problèmes de performance et de sécurité. Elle est néanmoins évoquée brièvement dans cette partie car vous pouvez la rencontrer dans du code PHP "ancien".

Si la directive de configuration `register_globals` est à `on`, toutes les zones de formulaire sont automatiquement enregistrées dans des variables dans le script PHP qui traite le formulaire. Les variables héritent du nom (attribut `name` de la balise `<input>`, `<select>` ou `<textarea>`) de la zone correspondante du formulaire. Ce comportement est valable quelle que soit la méthode (`GET` ou `POST`).

De même, toutes les données de l'URL sont automatiquement enregistrées dans des variables dans le script PHP qui traite l'URL. Les variables héritent du nom du paramètre correspondant de l'URL.

## d. Que choisir ?

Une des deux premières méthodes.

Les évolutions successives de PHP vont dans le sens d'une utilisation des tableaux `$_POST` ou `$_GET`, ou de la fonction `import_request_variables`. Dans une future version (la 5.3 ou 6), la directive de configuration `register_globals` sera définitivement passée à `off` et ne sera pas modifiable dans le fichier `php.ini`. Dans une telle hypothèse, la troisième méthode ne sera plus disponible.

La communauté PHP conseille vivement d'utiliser une des deux premières méthodes pour des raisons de sécurité : en interrogeant le tableau adéquat, ou en important explicitement les variables, on est certain que l'information vient bien de l'endroit prévu et qu'il n'y a pas eu substitution par un utilisateur mal intentionné.

Dans la suite de ce chapitre, nous allons présenter plus en détail l'utilisation de ces méthodes pour récupérer des données passées dans l'URL ou des données saisies dans un formulaire (utilisant la méthode `POST`).

---

Les tableaux `$_POST`, `$_GET` et `$_REQUEST` sont des tableaux superglobaux. Ils sont disponibles dans tous les contextes d'exécution.

---

Dans le chapitre Accéder à une base de données MySQL, après avoir vu d'autres tableaux similaires, nous ferons une petite synthèse sur les variables GPCS (Get/Post/Cookie/Session).

---

# Récupérer les données passées par l'URL

## 1. Considérations

### a. Que se passe-t-il si deux paramètres portent le même nom ?

C'est tout simplement le dernier paramètre rencontré dans l'URL qui fixe la valeur.

*Exemple*

```
<a href="page2.php?nom=Olivier&nom=Xavier">Page 2</a>
```

Cette URL donne une seule variable `nom` égale à `Xavier` dans le tableau `$_GET` ou lors de l'import avec `import_request_variables`.

### b. Utiliser un tableau pour passer des données dans l'URL

Il est possible d'utiliser une notation de type tableau dans le nom du paramètre passé dans l'URL.

*Exemple*

```
<a href="page2.php?data[]=HEURTEL&data[]=Olivier">Page 2</a>
```

Cette URL donne une variable `data`, de type tableau, qui contient les lignes suivantes :

| Clé | Valeur |
|---|---|
| 0 | HEURTEL |
| 1 | Olivier |

PHP remplit le tableau en ajoutant une ligne pour chaque paramètre, avec un indice entier consécutif commençant à 0 (comme pour la notation `[]` étudiée dans le chapitre Introduction à PHP - Les bases du langage PHP).

Cette technique est intéressante, mais, dans le code, il faut savoir que l'indice 0 correspond au nom et l'indice 1 au prénom. Par ailleurs, un problème peut se présenter si l'ordre des paramètres change.

Pour améliorer cette technique, il est possible de fixer soi même la clé, soit avec un numéro, soit avec une chaîne de caractères.

*Exemple*

```
<a href="page2.php?data[nom]=HEURTEL&data[prénom]=Olivier">
   Page 2</a>
```

Cette URL donne le résultat suivant dans le tableau `data` :

| Clé | Valeur |
|---|---|
| nom | HEURTEL |
| prénom | Olivier |

## 2. Transmettre des caractères spéciaux

Si la valeur à transmettre ne contient pas de caractères spéciaux (espace, éperluette (`&`), point d'interrogation (`?`), etc.), elle peut être placée directement dans l'URL comme indiqué précédemment. Dans le cas contraire, il est nécessaire de l'encoder pour éviter que ces caractères particuliers soient mal interprétés.

Par exemple, si la données passée dans l'URL contient "`Olivier & Xavier`", seul `Olivier` sera récupéré à l'arrivée car le `&` est interprété comme le séparateur de paramètres.

L'encodage nécessaire peut être réalisé très facilement grâce aux fonctions `urlencode` ou `rawurlencode`.

*Syntaxe*

```
chaîne urlencode(chaîne valeur)
chaîne rawurlencode(chaîne valeur)
```

valeur

Chaîne à encoder.

Ces deux fonctions retournent la chaîne après encodage. L'encodage consiste à remplacer tous les caractères non alphanumériques par une séquence %xy, xy étant un nombre hexadécimal égal au code ASCII du caractère. La différence entre les deux fonctions est subtile et concerne juste le caractère espace : la fonction urlencode remplace les espaces par le caractère plus (+), le vrai caractère "plus" étant lui même encodé, alors que la fonction rawurlencode remplace les espaces par la séquence %20 (code ASCII 32 en hexadécimal). La fonction urlencode est conforme au type MIME application/x-www-form-urlencoded (type utilisé pour transmettre les valeurs des formulaires) alors que la fonction rawurlencode est conforme à la RFC1738 ; a priori, il faut plutôt utiliser la fonction rawurlencode.

*Exemple*

```
<?php
// Initialisation d'une variable.
$nom='Olivier & Xavier';
echo urlencode($nom),'<br />';
echo rawurlencode($nom),'<br />';
?>
```

*Résultat*

```
Olivier+%26+Xavier
Olivier%20%26%20Xavier
```

*Exemple complet*

- Le script page1.php peut être modifié de la manière suivante :

```
<?php
// Initialisation d'une variable.
$nom='Olivier & Xavier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans lURL -->
    <a href=
         "page2.php?nom=<?php echo rawurlencode($nom); ?>">
         Page 2</a>
    </div>
  </body>
</html>
```

- Source de la page dans le navigateur

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur de $nom
         dans l'URL -->
    <a href="page2.php?nom=Olivier%20%26%20Xavier">
       Page 2</a>
    </div>
```

```
  </body>
</html>
```

- Script page2.php

```
<?php
// Affichage l'information passée dans l'URL
// (utilisation du tableau $_GET).
echo $_GET['nom'];
?>
```

- Résultat affiché à l'arrivée

```
Olivier & Xavier
```

La chaîne de la requête peut aussi être construite à l'aide de la fonction `http_build_query` apparue en version 5.

*Syntaxe*

```
chaîne http_build_query(tableau données [, chaîne préfixe])
```

`données`

Tableau contenant les données à utiliser pour construire la chaîne de la requête. L'indice ou la clé du tableau sont utilisés comme nom du paramètre pour la valeur associée.

`préfixe`

Préfixe à utiliser pour le nom des paramètres, lorsqu'il s'agit d'un indice numérique. Permet d'avoir à l'arrivée un nom exploitable comme nom de variable (un nom de variable PHP ne peut pas commencer par un chiffre).

Cette fonction construit, puis encode, une chaîne de requête sous la forme `clé1=valeur1&clé2=valeur2&...` en utilisant les clés (ou indices) et valeurs trouvées dans le tableau `données`. S'il est saisi, le paramètre `préfixe` est ajouté devant les indices numériques.

*Exemple*

```
<?php
// Initialisation du tableau contenant les données.
$données=array('nom' => 'Olivier & Xavier','David + Thomas');
// Construction de la chaîne de la requête :
// - sans préfixe
echo http_build_query($données),'<br />';
// - avec préfixe
echo http_build_query($données,'v_'),'<br />';
?>
```

*Résultat*

```
nom=Olivier+%26+Xavier&0=David+%2B+Thomas
nom=Olivier+%26+Xavier&v_0=David+%2B+Thomas
```

---

Il existe deux fonctions, `urldecode` et `rawurldecode`, qui permettent de décoder une chaîne préalablement encodée, respectivement par `urlencode` ou `rawurlencode`. Ces fonctions de décodage n'ont pas besoin d'être appelées lorsque des données encodées sont transmises par l'URL. En effet, ces données sont automatiquement décodées à l'arrivée.

---

# Récupérer les données saisies dans le formulaire

## 1. Considérations

### a. Que se passe-t-il si deux zones portent le même nom ?

C'est tout simplement la dernière zone rencontrée dans le formulaire qui fixe la valeur.

*Exemple*

```
<form action="saisie.php" method="POST"><div>
Nom : <input type="text" name="nom"><br />
Prénom : <input type="text" name="nom"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

La saisie de `HEURTEL` dans la première zone et de `Olivier` dans la deuxième donne une seule variable égale à `Olivier` dans le tableau `$_POST` ou lors de l'import avec `import_request_variables`.

### b. Que se passe-t-il s'il y a deux formulaires dans la page HTML ?

Les variables ne seront créées et renseignées que pour le formulaire qui a été validé.

*Exemple*

```
<form action="saisie.php" method="POST"><div>
Nom 1 : <input type="text" name="nom1"><br />
<input type="submit" name="ok1" value="OK1">
</div></form>
<form action="saisie.php" method="POST"><div>
Nom 2 : <input type="text" name="nom2"><br />
<input type="submit" name="ok2" value="OK2">
</div></form>
```

Si l'utilisateur valide le premier formulaire, la variable `nom1` sera disponible. Si l'utilisateur valide le deuxième formulaire, c'est la variable `nom2` qui sera disponible

### c. Utiliser un tableau pour récupérer les données saisies

Il est possible d'utiliser une notation de type tableau dans l'attribut `name` des balises `<input>`, `<select>` et `<textarea>`.

*Exemple*

```
<form action="saisie.php" method="POST"><div>
Nom : <input type="text" name="saisie[]"><br />
Prénom : <input type="text" name="saisie[]"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

La saisie de `HEURTEL` dans la première zone et de `Olivier` dans la deuxième donne une seule variable `saisie`, de type tableau, qui contient les lignes suivantes :

| Clé | Valeur |
|---|---|
| 0 | HEURTEL |
| 1 | Olivier |

PHP remplit le tableau en ajoutant une ligne pour chaque zone et avec un indice entier consécutif commençant à 0 (comme pour la notation `[]` étudiée dans le chapitre Introduction à PHP - Les bases du langage PHP).

Cette technique est intéressante, mais, dans le code, il faut savoir que l'indice 0 correspond au nom et l'indice 1 au prénom. Par ailleurs, un problème peut se présenter si l'ordre des zones change.

Pour améliorer cette technique, il est possible de fixer soi même la clé, soit avec un numéro, soit avec une chaîne de caractère.

*Exemple*

```
<form action="saisie.php" method="POST"><div>
Nom : <input type="text" name="saisie[nom]"><br />
Prénom : <input type="text" name="saisie[prénom]"><br />
<input type="submit" name="ok" value="OK">
</div></form>
```

La saisie de HEURTEL dans la première zone et de Olivier donne le résultat suivant dans le tableau saisie :

| Clé | Valeur |
|---|---|
| nom | HEURTEL |
| prénom | Olivier |

Nous verrons ultérieurement des situations où ce type de notation est obligatoire.

### d. Passer des informations dans une zone de formulaire cachée

Les informations saisies dans un formulaire sont transmises au script chargé du traitement et pouvant ensuite être affichées dans une nouvelle page.

Cette méthode peut être utilisée pour transmettre au passage d'autres informations non saisies par l'utilisateur, typiquement en les plaçant dans une zone de formulaire cachée.

*Exemple*

- Script page1.php

```
<?php
// Initialisation d'une variable.
$nom='Olivier & Xavier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 avec un bouton de formulaire -->
    <form action="page2.php" method="post">
    <!-- l'information à transmettre est cachée -->
    <input type="hidden" name="nom" value="<?php echo $nom; ?>" />
    <input type="submit" name="page2" value="Page 2" />
    </form>
    </div>
  </body>
</html>
```

- Source de la page dans le navigateur

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 avec un bouton de formulaire -->
    <form action="page2.php" method="post">
    <!-- l'information à transmettre est cachée -->
    <input type="hidden" name="nom" value="Olivier & Xavier" />
    <input type="submit" name="page2" value="Page 2" />
    </form>
    </div>
```

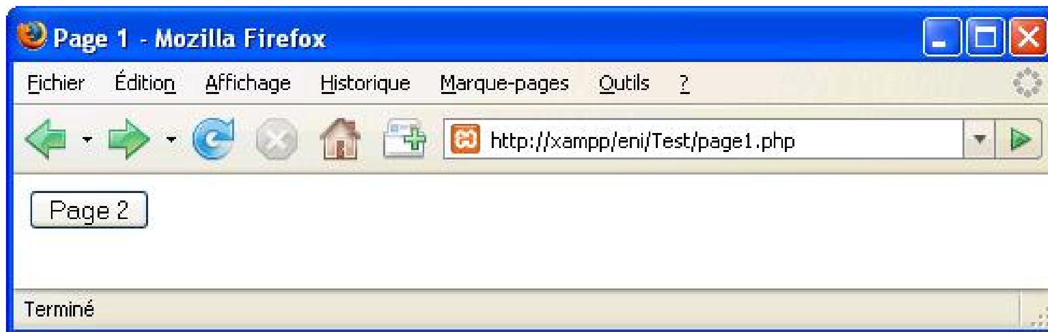
```
  </body>
</html>
```

- Script `page2.php`

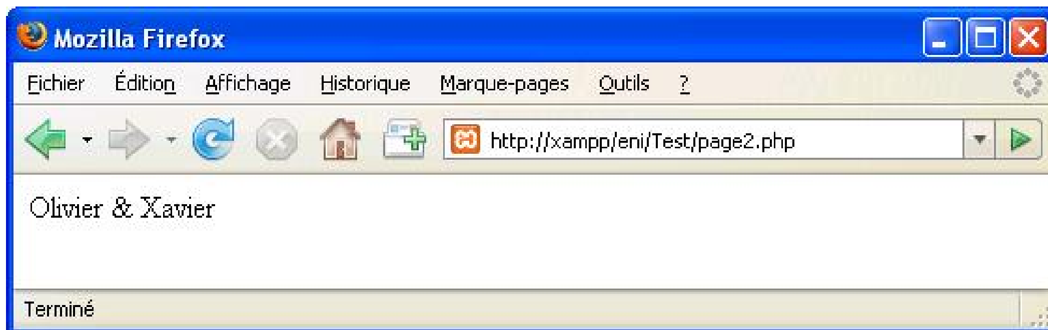
```
<?php
// Affichage de l'information cachée passée
// dans le formulaire.
echo $_POST['nom'];
?>
```

*Résultat*

- Affichage de la page 1




- Résultat du clic sur le bouton




## 2. Les différents types de zone

### a. Vue d'ensemble

Prenons notre formulaire complet de départ et voyons quelles sont les informations récupérées dans le script PHP.

*Script `saisie.htm`*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Saisie</title>
  </head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom" value=""
           size="20" maxlength="20" />
```

```
      Mot de passe :
      <input type="password" name="mot_de_passe" value=""
           size="20" maxlength="20" />
      <br />Sexe :
      <input type="radio" name="sexe" value="M" />Masculin
      <input type="radio" name="sexe" value="F" />Feminin
      <input type="radio" name="sexe" value="?"
                        checked="checked" />Ne sait pas
      <br />Photo :
      <input type="file" name="photo" value="" size="50" />
      <br />Couleurs préférées :
      <input type="checkbox" name="couleurs[bleu]" />Bleu
      <input type="checkbox" name="couleurs[blanc]" />Blanc
      <input type="checkbox" name="couleurs[rouge]" />Rouge
      <input type="checkbox" name="couleurs[pas]"
                            checked="checked" />Ne sait pas
      <br />Langue :
      <select name="langue">
        <option value="E">Espagnol</option>
        <option value="F" selected="selected" >Francais</option>
        <option value="I">Italien</option>
      </select>
      <br />Fruits préférés :<br />
      <select name="fruits[]" multiple="multiple" size="8">
        <option value="A">Abricots</option>
        <option value="C">Cerises</option>
        <option value="F">Fraises</option>
        <option value="P">Pêches</option>
        <option value="?" selected="selected">
                      Ne sait pas</option>
      </select>
      <br />Commentaire :<br />
      <textarea name="commentaire" rows="4" cols="50"></textarea>
      <br />
      <input type="hidden" name="invisible" value="123" /><br />
      <input type="submit" name="soumettre" value="OK" />
      <input type="image" name="valider" src="valider.gif" />
      <input type="reset" name="effacer" value="Effacer" />
      <input type="button" name="action" value="Ne fait rien" />
    </div>
    </form>
  </body>
</html>
```

*Script saisie.php*

```
<?php
// Inclusion d'un fichier contenant des fonctions génériques
// (dont la fonction afficher_tableau définie dans le
// chapitre 8)
include('fonctions.inc') ;
afficher_tableau($_POST,'$_POST :');
?>
```

*Résultat*

- Affichage initial et saisie de différentes valeurs :

Nom : HEURTEL Mot de passe : *******
Sexe : Masculin Feminin Ne sait pas
Photo : D:\Photos\identité.jpg Parcourir...
Couleurs préférées : Bleu Blanc Rouge Ne sait pas
Langue : Francais
Fruits préféres :
Abricots
Cerises
Fraises
Pêches
Ne sait pas
Commentaire :
Consultant en système d'information

OK Effacer Ne fait rien

- Résultat du clic sur le bouton **OK** (affichage du contenu de `$_POST`) :

```
$_POST :
nom = HEURTEL
mot_de_passe = olivier
sexe = M
photo = identité.jpg
couleurs =
  bleu = on
  rouge = on
langue = F
fruits =
  0 = A
  1 = F
commentaire = Consultant en système d'information
invisible = 123
soumettre = OK
```

Sur la base de cet exemple, nous allons apporter quelques explications.

### b. Zone contenant du texte

Pour les zones comportant du texte, c'est-à-dire les zones `<input>` de type `text`, `password`, `file` et `hidden`, ainsi que pour la zone `<textarea>`, les variables associées contiennent le texte saisi.

*Exemple*

```
$_POST :
nom = HEURTEL
mot_de_passe = olivier
photo = identité.jpg
commentaire = Consultant en système d'information
invisible = 123
```



Pour l'instant, avec le zone de type `file`, nous avons juste récupéré le nom du fichier, pas le fichier lui-même (cf. dans ce chapitre - Échanger un fichier entre le client et le serveur - Télécharger un fichier à partir du client :

"file upload").

---

### c. Groupe de boutons radio

Pour avoir un groupe de boutions radio, les zones doivent porter le même nom.

Pour un groupe de boutons radio, c'est-à-dire pour des zones `<input>` de type `radio`, la variable associée contient la valeur contenue dans l'attribut `value` de la balise `input` du bouton sélectionné. Si l'attribut `value` est absent, la valeur par défaut est `on`, ce qui est gênant puisqu'il est alors impossible de connaître l'option sélectionnée. Dans la pratique, il faut renseigner l'attribut `value`.

*Exemple*

```
<br />Sexe :
<input type="radio" name="sexe" value="M" /> Masculin
<input type="radio" name="sexe" value="F" /> Feminin
<input type="radio" name="sexe" value="?"
                        checked="checked" /> Ne sait pas
```

*Option sélectionnée*

Sexe : (•) Masculin ( ) Feminin ( ) Ne sait pas

*Résultat*

```
$_POST :
sexe = M
```

### d. Case à cocher

Pour des cases à cocher, c'est-à-dire pour des zones `<input>` de type `checkbox`, la variable associée contient la valeur contenue dans l'attribut `value` de la balise `input`. Si l'attribut `value` est absent, la valeur par défaut est `on`. Dans les deux cas, la variable associée est définie uniquement pour les cases cochées.

*Exemple*

```
<br />Couleurs preferees :
<input type="checkbox" name="bleu" value="b"/> Bleu
<input type="checkbox" name="blanc" /> Blanc
<input type="checkbox" name="rouge" /> Rouge
<input type="checkbox" name="nesaitpas"
                      checked="checked" />Ne sait pas
```

*Options sélectionnées*

Couleurs préférées : [x] Bleu [ ] Blanc [x] Rouge [ ] Ne sait pas

*Résultat*

```
$_POST :
bleu = b
rouge = on
```

Avec la case à cocher, l'attribut `value` a généralement moins d'importance, car il est possible de déterminer qu'une case est cochée uniquement par le fait que la variable associée a une valeur (peu importe cette valeur). Par contre, il est important que chaque case à cocher ait un nom différent.

Plusieurs approches sont possibles vis à vis de la valeur de l'attribut `value` :

- L'attribut `value` stocke la valeur souhaitée au niveau de la logique applicative dans le cas où la case est cochée (`1`, `oui` ...), sachant que, si la case n'est pas cochée, la variable n'existe pas.
- L'attribut `value` est omis et le code interprète l'existence de la variable selon les besoins de la logique applicative.

Dans le code, si vous souhaitez récupérer le fait d'une case soit cochée sous la forme d'un booléen, vous pouvez écrire une instruction du style :

```
$case_est_cochée = isset($_POST['nom_case'])?TRUE:FALSE;
```

Si vous le souhaitez, vous pouvez utiliser un tableau pour ne pas avoir une variable par case à cocher. Pour être en mesure de déterminer quelles sont les cases cochées, vous avez deux possibilités :

- Utiliser un tableau sans indice mais renseigner l'attribut value :

```
<br />Couleurs preferees :
<input type="checkbox" name="couleurs[]" value="bleu" /> Bleu
<input type="checkbox" name="couleurs[]" value="blanc" /> Blanc
<input type="checkbox" name="couleurs[]" value="rouge" /> Rouge
<input type="checkbox" name="couleurs[]" value="nesaitpas"
                             checked="checked" />Ne sait pas
```

- Ne pas renseigner l'attribut value mais définir soi même des indices ou des clés dans le tableau :

```
// indices numériques
<br />Couleurs preferees :
<input type="checkbox" name="couleurs[1]" /> Bleu
<input type="checkbox" name="couleurs[2]" /> Blanc
<input type="checkbox" name="couleurs[3]" /> Rouge
<input type="checkbox" name="couleurs[4]"
                             checked="checked" />Ne sait pas

// clés alphanumériques
<br />Couleurs preferees :
<input type="checkbox" name="couleurs[bleu]" /> Bleu
<input type="checkbox" name="couleurs[blanc]" /> Blanc
<input type="checkbox" name="couleurs[rouge]" /> Rouge
<input type="checkbox" name="couleurs[nesaitpas]"
                             checked="checked" />Ne sait pas
```

Avec ce dernier exemple, si les cases "Bleu" et "Rouge" sont cochées, le tableau couleur contiendra les lignes suivantes :

```
$_POST :
couleurs =
  bleu = on
  rouge = on
```

Si un tableau est utilisé pour l'ensemble du formulaire, vous pouvez utiliser différentes solutions parmis lesquelles :

- Utiliser un tableau sans indice mais renseigner l'attribut value :

```
<br />Couleurs preferees :
<input type="checkbox" name="saisie[couleurs][]" value="bleu" /> Bleu
<input type="checkbox" name="saisie[couleurs][]" value="blanc" /> Blanc
<input type="checkbox" name="saisie[couleurs][]" value="rouge" /> Rouge
<input type="checkbox" name="saisie[couleurs][]" value="nesaitpas"
                             checked="checked" />Ne sait pas
```

- Ne pas renseigner l'attribut value mais définir soi même des indices ou des clés dans le tableau :

```
// indices numériques
<br />Couleurs preferees :
<input type="checkbox" name="saisie[couleurs][1]" /> Bleu
<input type="checkbox" name="saisie[couleurs][2]" /> Blanc
<input type="checkbox" name="saisie[couleurs][3]" /> Rouge
<input type="checkbox" name="saisie[couleurs][4]"
                             checked="checked" />Ne sait pas

// clés alphanumériques
```

```
<br />Couleurs preferees :
<input type="checkbox" name="saisie[couleurs][bleu]" /> Bleu
<input type="checkbox" name="saisie[couleurs][blanc]" /> Blanc
<input type="checkbox" name="saisie[couleurs][rouge]" /> Rouge
<input type="checkbox" name="saisie[couleurs][nesaitpas]"
                            checked="checked" />Ne sait pas
```

Avec ce dernier exemple, si les cases "Bleu" et "Rouge" sont cochées, le tableau `saisie` contiendra les lignes suivantes :

```
$_POST :
saisie =
  couleurs =
    bleu = on
    rouge = on
```

### e. Liste à sélection unique

Pour les listes à sélection unique, c'est-à-dire pour une zone `<select>` sans attribut multiple, la variable associée contient la valeur contenue dans l'attribut `value` de la balise `<option>`, ou, s'il n'y a pas d'attribut `value`, la valeur affichée dans la liste (derrière la balise `<option>`).

*Exemple (avec attribut `value`)*

```
<br />Langue :
<select name="langue">
  <option value="E">Espagnol</option>
  <option value="F" selected="selected" >Francais</option>
  <option value="I">Italien</option>
</select>
```

*Option sélectionnée*

Langue : Francais

*Résultat*

```
$_POST :
langue = F
```

*Exemple (sans attribut `value`)*

```
<br />Langue :
<select name="langue">
  <option>Espagnol</option>
  <option selected="selected" >Francais</option>
  <option>Italien</option>
</select>
```

*Option sélectionnée*

Langue : Francais

*Résultat*

```
$_POST :
langue = Français
```

L'une ou l'autre des possibilités peut être choisie en fonction des besoins de la logique applicative. Souvent, l'option `value` est utilisée pour mettre un code qui sera stocké dans la base à la place de la valeur affichée. Cette approche présente l'inconvénient suivant : la partie interface utilisateur (couche présentation) doit connaître les codes, ce qui n'est pas une bonne idée. La solution optimale consiste alors à générer dynamiquement le formulaire à partir de la base.

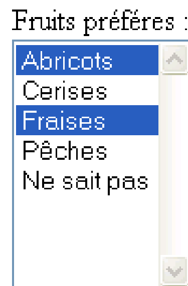
### f. Liste à sélection multiple

Pour les listes à sélection multiple, c'est-à-dire pour une zone `<select>` avec attribut `multiple`, la variable associée contient la valeur contenue dans l'attribut `value` de la balise `<option>`, ou, s'il n'y a pas d'attribut `value`, la valeur affichée dans la liste (derrière la balise `<option>`). Attention, ceci est valable uniquement pour la dernière option sélectionnée, si la variable est une variable scalaire. En conséquence, pour une liste à sélection multiple, il faut obligatoirement utiliser un tableau.

*Exemple (avec attribut value)*

```
<br />Fruits préférés :<br />
<select name="fruits[]" multiple="multiple" size="8">
  <option value="A">Abricots</option>
  <option value="C">Cerises</option>
  <option value="F">Fraises</option>
  <option value="P">Pêches</option>
  <option value="?" selected="selected">
                    Ne sait pas</option>
</select>
```

*Options sélectionnées*




*Résultat*

```
$_POST :
fruits =
  0 = A
  1 = F
```

Comme nous l'avons déjà vu, PHP remplit le tableau avec une ligne pour chaque option sélectionnée et numérote lui-même les lignes. Dans le cas de la liste à sélection multiple, ce mode de fonctionnement ne pose pas de problème.

*Exemple (sans option value)*

```
<br />Fruits préférés :<br />
<select name="fruits[]" multiple="multiple" size="8">
  <option>Abricots</option>
  <option>Cerises</option>
  <option>Fraises</option>
  <option>Pêches</option>
  <option selected="selected">
                    Ne sait pas</option>
</select>
```

*Résultat (avec la même sélection que précédemment)*

```
$_POST :
fruits =
  0 = Abricots
  1 = Fraises
```

L'une ou l'autre des possibilités peut être choisie en fonction des besoins de la logique applicative (même principes que pour la liste à sélection unique).

Utiliser un tableau pour l'ensemble du formulaire ne pose pas de problème.

*Exemple (avec attribut value)*

```
<br />Fruits préférés :<br />
<select name="saisie[fruits][]" multiple="multiple" size="8">
  <option value="A">Abricots</option>
  <option value="C">Cerises</option>
  <option value="F">Fraises</option>
  <option value="P">Pêches</option>
  <option value="?" selected="selected">
                    Ne sait pas</option>
</select>
```

*Résultat (avec la même sélection que précédemment)*

```
$_POST :
saisie =
  fruits =
    0 = A
    1 = F
```

### g. Bouton de validation

Pour un bouton de validation, c'est-à-dire pour une zone <input> de type submit, PHP crée une variable portant le nom du bouton (attribut name) et ayant comme valeur, la valeur de l'attribut value, uniquement si le bouton est cliqué.

*Exemple*

```
<input type="submit" name="soumettre" value="OK" />
```

*Résultat (si le bouton est cliqué)*

```
$_POST :
soumettre = OK
```

Si le bouton ne porte pas de nom, aucune variable n'est créée. Ce n'est pas grave si :

- Il n'est pas nécessaire de savoir comment le script est appelé (affichage initial ou traitement du formulaire).
- Il n'y a qu'un seul bouton de validation.

Dans les autres cas, il faut nommer le(s) bouton(s).

Pour faire la différence entre l'appel du script pour affichage initial et l'appel du script pour traiter le formulaire (cf. dans ce chapitre - Vue d'ensemble - Les formulaires), il suffit de tester l'existence de la variable, par exemple avec la fonction PHP isset.

*Exemple*

```
<?php
...
// Tester comment le script est appelé
if (isset($_POST['soumettre'])) {
  // Une ligne existe dans la variable $_POST
  // correspondant au bouton OK nommé « soumettre » :
  // le script est appelé sur la validation du formulaire.
  // => Traiter le formulaire ...
  ...
} else {
  // Le script n'est pas appelé par le clic sur le
  // bouton OK. S'il n'y pas d'autre bouton "submit", le
  // script est donc appelé pour l'affichage initial.
  // => Initialiser le formulaire ...
  ...
}
?>
```

Un problème se pose si le formulaire possède une seule zone de texte, aucun bouton et que l'utilisateur tape ENTER ou RETURN. Dans ce cas, le formulaire est bien soumis mais il n'existe pas de bouton de validation pour réaliser le test dans le script PHP. La solution consiste à tester si la variable $_POST (ou $_GET, ou $_REQUEST) est vide ou pas (avec

empty mais pas isset car le tableau existe toujours).

Si le formulaire contient deux boutons de validation nommés différemment (attribut name), le premier ok et le deuxième annuler, il est possible de déterminer dans quel contexte le script est appelé.

*Exemple*

```
// Tester comment le script est appelé
if (isset($_POST['ok'])) {
     // bouton OK
} elseif (isset($_POST['annuler'])) {
     // bouton Annuler
} else {
     // affichage initial
}
```

Si le formulaire contient deux boutons de validation portant le même nom (attribut name="soumettre" par exemple), mais des valeurs (attribut value) différentes, le premier OK et le deuxième Annuler, il est possible de déterminer dans quel contexte le script est appelé.

*Exemple*

```
// Tester comment le script est appelé
if ($_POST['soumettre'] == 'OK') {
      // bouton OK
} elseif ($_POST['soumettre'] == 'Annuler') {
      // bouton Annuler
} else {
      // affichage initial
}
```

### h. Bouton image

Pour un bouton image, c'est-à-dire pour une zone <input> de type image, PHP crée deux variables portant le nom du bouton (attribut name) suivi de _x et _y, et donnant la position relative, en pixels, du clic par rapport au coin haut gauche de l'image (uniquement si l'image est cliquée). Si le bouton ne porte pas de nom, les deux variables sont nommées x et y.

*Exemple*

```
<input type="image" name="valider" src="valider.gif" />
```

*Résultat (si l'image est cliquée)*

```
$_POST :
valider_x = 5
valider_y = 8
```

Il est alors possible de traiter la position du clic si elle est significative du point de vue de la logique applicative.

Si plusieurs boutons images sont présents dans le formulaire, il est possible de déterminer quel bouton a provoqué la soumission du formulaire.

```
// Tester comment le script est appelé
if (isset($_POST['valider_x'])) {
      // bouton Valider
} else ...
```

### i. Bouton "reset" ou "button"

Un clic sur des boutons correspondant à des zones <input> de type reset ou button ne provoque pas la soumission du formulaire, ni l'appel du script de traitement. Ces boutons permettent de faire une simple action côté navigateur (en JavaScript par exemple).

## 3. Synthèse

Il faut prendre l'habitude de bien nommer (attribut name) toutes les zones du formulaire avec des noms distincts ou d'utiliser un nommage de type tableau, plutôt associatif, pour faciliter la maintenance et la lisibilité du code.

Pour les groupes de boutons radio, il faut mettre un attribut value distinct à chaque bouton.

Pour les cases à cocher, il faut utiliser des noms distincts (attribut name), et/ou des valeurs distinctes (attribut value), pour être certain de pouvoir faire la différence à l'arrivée ; en cas d'utilisation d'un tableau ne laissez pas PHP effectuer la numérotation (pas de []).

Pour les listes à sélection multiple, il faut utiliser un nommage de type tableau pour pouvoir récupérer la liste des valeurs sélectionnées ; employez l'attribut value pour récupérer une valeur différente de celle qui apparaît dans la liste.

De même, Il faut nommer le(s) bouton(s) de validation pour être en mesure de savoir comment le script PHP est appelé. Si plusieurs boutons sont utilisés, il est possible d'employer le même nom du moment que les valeurs (attribut value) sont différentes.

Le formulaire utilisé tout au long de ce sous-titre 2 est un bon exemple de l'application de ces différentes recommandations.

*Même exemple avec un tableau*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Saisie</title>
  </head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="saisie[nom]" value=""
           size="20" maxlength="20" />
      Mot de passe :
      <input type="password" name="saisie[mot_de_passe]" value=""
           size="20" maxlength="20" />
      <br />Sexe :
      <input type="radio" name="saisie[sexe]" value="M" /> Masculin
      <input type="radio" name="saisie[sexe]" value="F" /> Feminin
      <input type="radio" name="saisie[sexe]" value="?"
                         checked="checked" /> Ne sait pas
      <br />Photo :
      <input type="file" name="saisie[photo]" value="" size="50" />
      <br />Couleurs préférées :
      <input type="checkbox" name="saisie[couleurs][bleu]" />Bleu
      <input type="checkbox" name="saisie[couleurs][blanc]" />Blanc
      <input type="checkbox" name="saisie[couleurs][rouge]" />Rouge
      <input type="checkbox" name="saisie[couleurs][pas]"
                             checked="checked" />Ne sait pas
      <br />Langue :
      <select name="saisie[langue]">
        <option value="E">Espagnol</option>
        <option value="F" selected="selected" >Francais</option>
        <option value="I">Italien</option>
      </select>
      <br />Fruits préférés :<br />
      <select name="saisie[fruits][]" multiple="multiple" size="8">
        <option value="A">Abricots</option>
        <option value="C">Cerises</option>
        <option value="F">Fraises</option>
        <option value="P">Pêches</option>
        <option value="?" selected="selected">
                  Ne sait pas</option>
      </select>
      <br />Commentaire :<br />
      <textarea name="saisie[commentaire]"
                  rows="4" cols="50"></textarea>
      <br />
      <input type="hidden" name="saisie[invisible]" value="123" />
      <br />
```

```
      <input type="submit" name="soumettre" value="OK" />
      <input type="image" name="valider" src="valider.gif" />
      <input type="reset" name="effacer" value="Effacer" />
      <input type="button" name="action" value="Ne fait rien" />
    </div>
    </form>
  </body>
</html>
```

*Résultat (avec la même saisie que dans l'exemple initial)*

```
$_POST :
saisie =
  nom = HEURTEL
  mot_de_passe = olivier
  sexe = M
  photo = identité.jpg
  couleurs =
    bleu = on
    rouge = on
  langue = F
  fruits =
    0 = A
    1 = F
  commentaire = Consultant en système d'information
  invisible = 123
soumettre = OK
```

# Contrôler les données récupérées

## 1. Vue d'ensemble

Dans la première partie de ce chapitre, nous avons vu comment récupérer les données passées dans une URL ou saisies dans un formulaire.

Ensuite, il faut vérifier que les données récupérées sont correctes, c'est-à-dire qu'elles respectent les règles de gestion définies pour l'application.

Pour la sécurité du site, il convient d'être suspicieux vis-à-vis des données qui viennent de l'extérieur (formulaire, URL, mais aussi nous le verrons plus tard, cookie, etc.). Ces données doivent être contrôlées, filtrées, pour éviter les attaques potentielles d'un utilisateur mal intentionné.

L'objectif de cette partie est de vous fournir quelques pistes sur les techniques couramment utilisées en PHP pour cette vérification. Dans le cas d'un formulaire, une autre approche possible consiste à réaliser un contrôle en JavaScript dans le navigateur, l'intérêt étant d'éviter un aller-retour avec le serveur.

Les différents exemples présentés dans cette partie utilisent des formulaires. Les mêmes vérifications doivent être effectuées sur les données passées dans une URL.

## 2. Vérifications classiques

### a. Nettoyage des espaces qui traînent

La fonction `trim` (cf. chapitre Utiliser les fonctions PHP - Manipuler les chaînes de caractères) peut être utilisée pour supprimer les "blancs" indésirables en début et/ou en fin de chaîne. Dans le cas d'un formulaire, ce traitement s'applique notamment aux zones en saisie libre (`<input>` de type `text` ou `password`, `<textarea>`).

*Exemple*

```
// Récupérer la valeur saisie dans la zone "nom" et nettoyer
// les espaces qui traînent (au début et à la fin)
$nom = trim($_POST['nom']);
```

### b. Donnée obligatoire

Tester si une donnée obligatoire est présente se révèle très simple : il suffit de tester si la variable associée contient une valeur.

*Exemple*

```
$nom = trim($_POST['nom']);
if ($nom == '') {
  // $nom vide = zone "nom" non saisie => faire quelque chose
}
```

### c. Longueur maximum d'une chaîne

La longueur des données récupérées peut être contrôlée avec la fonction `strlen` (cf. chapitre Utiliser les fonctions PHP - Manipuler les chaînes de caractères). Dans le cas d'un formulaire, ce traitement s'applique notamment aux zones en saisie libre (`<input>` de type `text` ou `password`, `<textarea>`).

*Exemple*

```
$nom = trim($_POST['nom']);
if (strlen($nom) > 20) {
  // $nom trop long => faire quelque chose
}
```

Dans un formulaire, l'attribut `maxlength` de la balise `input` permet de faire un contrôle complémentaire au moment de la saisie (c'est le navigateur qui s'en charge).

## d. Caractères autorisés pour une chaîne - Format

En cas de besoin, l'utilisation des expressions régulières (cf. chapitre Utiliser les fonctions PHP - Manipuler les chaînes de caractères) permet de contrôler très facilement que seuls certains caractères sont présents et éventuellement que la chaîne saisie respecte un format spécifique.

À titre d'exemple, supposons qu'un mot de passe doivent vérifier la règle suivante : commencer par une lettre, suivie de lettres, de chiffres ou des caractères _#*$ avec une longueur minimale de 4.

*Exemple*

```
$mot_de_passe = trim($_POST['mot_de_passe']);
if (! eregi('^[a-z][a-z0-9_#*$]{3,}',$mot_de_passe)) {
  // Mot de passe non valide.
}
```

Quelques explications sur l'expression régulière utilisée (`^[a-z][a-z0-9_#*$]{3,}`) :

- `eregi` autorise indifféremment les majuscules et minuscules
- `^` = commence par ...
- `[a-z]` = une lettre entre `a` et `z` (ou `A` et `Z` car `eregi` est utilisée) ...
- `[a-z0-9_#*$]{3,}` = suivi d'au moins 3 (`{3,}`) caractères parmi ceux indiqués : `a` à `z` (et donc `A` à `Z`), `0` à `9` et les caractères `_#*$`

D'autres exemples seront présentés dans la suite avec les dates et les nombres.

## e. Validité d'une date - Plage de valeurs

Généralement, deux vérifications doivent être faites sur une date

- respect d'un format (JJ/MM/AAAA par exemple) ;
- validité (pas de 32/13/2001).

La vérification du respect du format et des caractères autorisés est très simple avec une expression régulière.

*Exemple*

```
$date_naissance = trim($_POST['date_naissance']);
$format_date = '^[0-9]{1,2}/[0-9]{1,2}/[0-9]{4}$';
if (! ereg($format_date,$date_naissance)) {
  // Mauvais format pour la date.
}
```

Quelques explications sur l'expression régulière utilisée (`^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$`) :

- `^` = commence par ...
- `[0-9]{1,2}` = un ou deux chiffres ...
- `/` = suivi du caractère "/" ...
- `[0-9]{1,2}` = un ou deux chiffres ...

- / = suivi du caractère "/" ...

- [0-9]{4} = suivi de 4 chiffres ...

- $ = suivi de ... rien du tout. La chaîne doit se terminer immédiatement.

Pour vérifier la validité de la date, il est possible d'utiliser la fonction checkdate (cf. chapitre Utiliser les fonctions PHP - Manipuler les dates).

Dans un premier temps, il faut récupérer les composantes de la date saisie. Plusieurs méthodes sont applicables :

- avec la fonction explode :

```
$jma = explode('/',$date_naissance);
// $jma[0] contient le jour
// $jma[1] contient le mois
// $jma[2] contient l'année
if (! checkdate($jma[1],$jma[0],$jma[2])) {
  // Date non valide.
}
```

- variante avec les fonctions explode et list :

```
list($jour,$mois,$année) = explode('/',$date_naissance);
// Récupération des composantes dans des variables
// indépendantes.
if (! checkdate($mois,$jour,$année)) {
  // Date non valide.
}
```

- avec le troisième paramètre de la fonction ereg qui permet de récupérer des portions de la chaîne, et cela directement dans le test de conformité du format :

```
$format_date = '^([0-9]{1,2})/([0-9]{1,2})/([0-9]{4})$';
if (! ereg($format_date,$date_naissance,$jma)) {
  // Mauvais format pour la date .
} else {
  // Bon format pour la date
  // $jma contient les différentes composantes avec des
  // indices différents par rapport à l'exemple précédent :
  // $jma[1] contient le jour
  // $jma[2] contient le mois
  // $jma[3] contient l'année
  if (! checkdate($jma[2],$jma[1],$jma[3])) {
    // Date non valide.
  }
}
```

Pour tester la plage de valeurs, le plus simple est de faire une comparaison numérique (ou alphabétique) sur un nombre (une chaîne) construit sur la forme AAAAMMJJ (20071211 pour le 11/12/2007).

_Exemple_

```
// Récupérer les composantes de la date.
list($jour,$mois,$année) = explode('/',$date_naissance);
// Contruire une chaîne au format AAAAMMJJ
$aaaammjj = sprintf('%04d%02d%02d',$année,$mois,$jour);
// Définir les dates mini et maxi selon le même format.
$date_mini = '19000101'; // 01/01/1900
$date_maxi = date('Ymd'); // date du jour
// Comparer.
if ($aaaammjj < $date_mini or $aaaammjj > $date_maxi) {
  // Date hors de la plage autorisée
}
```

### f. Validité d'un nombre - Plage de valeurs

Plusieurs techniques peuvent être utilisées pour vérifier qu'un nombre est bien formé :

- les expressions régulières ;
- la fonction `is_numeric` ;
- la conversion de la saisie et le test du résultat obtenu.

*Exemples avec les expressions régulières*

```
// Vérifier qu'une donnée est un nombre entier.
ereg('^[+|-]?[0-9]+$',$variable)
// Vérifier qu'une donnée est un nombre entier et contrôler
// le nombre de chiffres (entre 1 et 3 par exemple)
ereg('^[+|-]?[0-9]{1,3}$',$variable)
// Vérifier qu'une donnée est un nombre décimal (sur cet
// exemple la virgule et le point sont acceptés comme
// séparateur décimal).
ereg('^[+|-]?[0-9]+[.|,]?[0-9]*$',$variable)
```

Pour la plage de valeurs, un simple test du type suivant suffit.

*Exemple*

```
if ($variable < minimum or $variable > maximum) {
  // $variable en dehors de la plage autorisée
}
```

### g. Validité d'une adresse e-mail

Là encore, les expressions régulières vont être utiles. De nombreux exemples peuvent être trouvés sur Internet.

La solution suivante fonctionne correctement pour les structures d'adresses courantes :

```
eregi(
  '^[a-z0-9]+([._-]?[a-z0-9]+)*'.             // début
  '@'.                                         // suite
  '[a-z0-9]+([.-]?[a-z0-9]+)*\.[a-z]{2,4}$', // fin
  $adresse_mail
)
```

Pour faciliter la lecture, l'expression régulière est construite par concaténation de trois chaînes.

Quelques explications sur l'expression régulière utilisée :

- `eregi` pour autoriser indifféremment les majuscules et les minuscules.
- ^ = commence par ...
- `[0-9a-z]+` = une séquence comportant des lettres ou des chiffres ...
- `([._-]?[a-z0-9]+)*` = suivi éventuellement d'une ou plusieurs séquences commençant chacune optionnellement par un tiret, un souligné ou un point suivi de lettres ou de chiffres ...
- `@` = suivi d'un @ ...
- `[0-9a-z]+` = suivi d'une séquence comportant des lettres ou des chiffres ...
- `([.-]?[a-z0-9]+)*` = suivi éventuellement d'une ou plusieurs séquences commençant chacune optionnellement par un tiret ou un point suivi de lettres ou de chiffres ...

- `\.` = suivi d’un point ...
- `[a-z]{2,3}` = suivi de deux à quatre lettres ...
- `$` = suivi de ... rien du tout. La chaîne doit se terminer.

---

L’expression régulière permet de vérifier que l’adresse est syntaxiquement correcte, mais pas de contrôler qu’elle existe réellement.

---

# Les problèmes sur les données récupérées

## 1. La fonctionnalité de "magic quotes"

Comme nous l'avons vu dans le chapitre Utiliser les fonctions PHP - Gérer les "guillemets magiques" ("magic quotes"), PHP propose une fonctionnalité, appelée "magic quotes" ("guillemets magiques") qui protège les données saisies dans un formulaire ou transmises dans une URL.

Pour mémoire, si la directive de configuration `magic_quotes_gpc` est à `on`, toutes les données arrivant par une méthode `GET` ou `POST` sont automatiquement protégées avec un anti slash (\) devant les caractères apostrophe ('), guillemet ("), anti slash (\) et nul (`\0`).

Considérons le script PHP `saisie.php` suivant qui se contente d'afficher une zone de saisie puis de réafficher ensuite la valeur saisie dans la zone du formulaire et directement dans la page.

```
<?php
// Récupérer la donnée saisie.
$saisie = isset($_POST['saisie'])?$_POST['saisie']:'';
// La réafficher ensuite dans la zone de saisie et
// directement dans la page.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Saisie : <input type="text" name="saisie"
                      value="<?php echo $saisie; ?>" />
      <input type="submit" name="ok" value="OK" />
      <br /><?php echo $saisie; ?>
    </div>
    </form>
  </body>
</html>
```

*Résultat*

- Affichage initial :

Saisie : [ ] OK

- Saisie d'une valeur :

Saisie : l'été \ l'hiver OK

- Résultat après un clic sur le bouton **OK** si `magic_quotes_gpc = on` :

Saisie : l\'été \\ l\'hiver OK
l\'été \\ l\'hiver

- Résultat après un clic sur le bouton **OK** si `magic_quotes_gpc = off` :

Saisie : l'été \ l'hiver OK
l'été \ l'hiver

Un problème similaire peut se poser avec les données transmises dans l'URL.

*Exemple*

- Script page1.php :

```
<?php
// Initialisation d'une variable.
$data="l'été \ l'hiver";
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <!-- lien vers la page 2 en passant la valeur
         dans l'URL -->
    <a href=
        "page2.php?data=<?php echo rawurlencode($data); ?>">
        Page 2</a>
    </div>
  </body>
</html>
```

- Script page2.php :

```
<?php
// Affichage l'information passée dans l'URL.
echo $_GET['data'];
?>
```

- Résultat après un clic sur le lien si magic_quotes_gpc = on :

```
l\'été \\ l\'hiver
```

Pour se conformer à la stratégie qui consiste à ne pas avoir de donnée protégée dans les variables (cf. chapitre Utiliser les fonctions PHP - Gérer les "guillemets magiques" ("magic quotes")), il faut systématiquement appeler une fonction lors de la récupération d'une donnée GPC pour enlever la protection "magic quotes" si cette dernière est active.

*Exemple*

```
<?php
function supprimer_encodage_MQ($valeur) {
  // Si magic quotes est actif, retourner la valeur après
  // la valeur après suppression de l'encodage
  // (=> appel à stripslashes), sinon, retourner
  // la valeur.
  return
    (get_magic_quotes_gpc())?stripslashes($valeur):$valeur;
}
?>
```

Appelez cette fonction uniquement sur des données GPC pour ne pas risquer d'enlever des caractères d'échappement qui ne sont pas liés à une protection "magic quotes".

La récupération des données d'un formulaire prend alors l'allure suivante :

```
<?php
// Inclusion du fichier qui contient les définitions de nos
// fonctions générales
include('fonctions.inc');
// Récupération des valeurs saisies dans le formulaire.
$nom = supprimer_encodage_MQ(trim($_POST['nom']));
$mot_de_passe =
  supprimer_encodage_MQ(trim($_POST['mot_de_passe']));
$date_naissance =
```

```
  supprimer_encodage_MQ(trim($_POST['date_naissance']));
// ...
?>
```

Pour les zones qui ne sont pas en saisie libre (liste à sélection unique ou multiple, groupe de boutons radio, cases à cocher), il n'est pas forcément nécessaire d'appeler la fonction car la donnée retournée est maîtrisée et vous pouvez faire en sorte qu'elle ne contienne pas d'apostrophe ; dans le doute, appelez la systématiquement (en l'adaptant au cas où la variable est un tableau).

Un appel similaire peut être effectué lors de la récupération des données transmises dans une URL :

```
<?php
// Affichage l'information passée dans l'URL.
$nom = supprimer_encodage_MQ($_GET['nom']);
?>
```

Vous pouvez aussi écrire une fonction qui supprime systématiquement et en un seul appel la protection "magic quotes" de tous les tableaux GPC.

*Exemple*

```
function  supprimer_encodage_MQ_GPC() {
  // Si magic quotes est actif ...
  if (get_magic_quotes_gpc()) {
    // Définir une fonction "strip_slashes" qui accepte un
    // paramètre par référence (important) et qui lui
    // applique la fonction "stripslashes".
    function strip_slashes(&$valeur) {
      $valeur = stripslashes($valeur);
    }
    // Appliquer cette fonction "strip_slashes" de façon
    // récursive à tous les tableaux GPC.
    array_walk_recursive($_POST, 'strip_slashes');
    array_walk_recursive($_GET, 'strip_slashes');
    array_walk_recursive($_REQUEST, 'strip_slashes');
    array_walk_recursive($_COOKIE, 'strip_slashes');
  }
}
```

La récupération des données saisies dans un formulaire ou transmises dans une URL prend alors l'allure suivante :

```
<?php
// Inclusion du fichier qui contient les définitions de nos
// fonctions générales.
include('fonctions.inc');
// Appel de la fonction qui supprime l'encodage
// "magic quotes" de tous les tableaux GPC.
supprimer_encodage_MQ_GPC();
// Récupération des valeurs saisies dans le formulaire.
$nom = trim($_POST['nom']);
$mot_de_passe = trim($_POST['mot_de_passe']);
$date_naissance = trim($_POST['date_naissance']);
// ...
?>
```

Cette deuxième solution présente deux avantages :

- toutes les données GPC sont traitées en un seul appel ;
- la migration vers la version de PHP qui désactivera définitivement la fonctionnalité "magic quotes" sera facilitée (une seule ligne de code à supprimer).

## 2. Autres problèmes sur les données externes

Un autre problème d'affichage dans une zone de formulaire peut se produire si la donnée affichée contient un guillemet (").

*Exemple*

Saisie : il dit : "bonjour !" OK

Saisie : il dit : OK
il dit : "bonjour !"

*Source de la page dans le navigateur (extrait)*

```
<form action="saisie.php" method="post">
<div>
  Saisie : <input type="text" name="saisie"
                value="il dit : "bonjour !"" />
  <input type="submit" name="ok" value="OK" />
  <br />il dit : "bonjour !"    </div>
</form>
```

En HTML, dans les attributs (`value`, `name` ...) des balises, le délimiteur de chaîne est le guillemet. Dans l'attribut `value`, la séquence `"il dit : "` est considérée comme la valeur de l'attribut et le reste de la chaîne est ignoré. Le problème se produit même si le guillemet est échappé par le caractère \ car ce dernier n'est pas un caractère d'échappement en HTML.

Un autre problème d'affichage se produit dans la page si la donnée affichée contient des balises HTML.

*Exemple*

Saisie : Olivier <b>Heurtel OK
Olivier **Heurtel**

La saisie `Olivier <b>Heurtel` donne le mot `Heurtel` en gras lors de l'affichage dans la page HTML. La séquence `<b>` saisie par l'utilisateur se retrouve telle qu'elle dans le code source de la page et est donc interprétée par le navigateur comme la balise de mise en gras.

Enfin, nous pouvons rencontrer un troisième problème lors de la saisie dans une zone de commentaire.

*Exemple*

Saisie :
Première ligne.
Deuxième ligne.
Troisième ligne.
OK
Première ligne. Deuxième ligne. Troisième ligne.

Une saisie sur plusieurs lignes dans la zone `<textarea>` est bien réaffichée telle quelle dans la zone mais est affichée sans les sauts de lignes dans la page HTML. Le texte est bien présent avec des sauts de lignes dans le code source de la page, mais le saut de ligne, en dehors d'une zone `<textarea>`, n'est pas interprété par le navigateur : il faut mettre une balise `<br />`.

Nous voyons donc apparaître trois problèmes relatifs à l'affichage dans la page HTML de données en provenance d'une source externe (formulaire, URL, etc.) :

- présence du caractère guillemet qui peut poser un problème lors de l'utilisation de la donnée dans un formulaire (attribut `value`) ;
- présence de balises HTML valides qui sont interprétées comme telles par le navigateur :
- non prise en compte des sauts de lignes présents dans un texte.

Le deuxième "problème" peut présenter un intérêt dans les situations où vous souhaitez offrir la possibilité à un utilisateur de saisir un texte avec un peu de mise en forme pour un affichage ultérieur dans une page.

Pour résoudre ces trois problèmes, PHP propose quatre fonctions : `htmlspecialchars`, `htmlentities`, `nl2br` et `strip_tags`.

La fonction htmlspecialchars prend une chaîne de caractères et la retourne en remplaçant certains caractères par leur équivalent HTML :

*Syntaxe*

```
chaîne htmlspecialchars(chaîne texte [, entier option [,
chaîne jeu [, booléen double_encodage]]])
```

texte

Chaîne à traiter.

option

Fonctionnement vis-à-vis des caractères guillemets (") et apostrophe (’).

jeu

Jeu de caractères à utiliser pour la conversion.

double_encodage

Indique s’il faut (TRUE, valeur par défaut) ou non (FALSE) encoder les entités HTML déjà encodées. Apparu en version 5.2.3.

Les caractères convertis sont les suivants :

| Entrée | Sortie |
|---|---|
| & | & |
| " | " |
| ’ | ' |
| < | < |
| > | > |

Le deuxième paramètre permet de préciser le fonctionnement vis à vis des caractères guillemets (") et apostrophe (’) :

| Valeur | Comportement |
|---|---|
| ENT_COMPAT | Conversion du guillemet mais pas de l’apostrophe (par défaut) |
| ENT_NOQUOTES | Aucune conversion |
| ENT_QUOTES | Conversion des deux caractères |

Le troisième paramètre permet de définir le jeu de caractères à utiliser pour la conversion : **ISO-8859-1** (par défaut), **ISO-8859-15**, **UTF-8**, etc.

Depuis la version 5.1.0, il existe une fonction htmlspecialchars_decode qui effectue la conversion inverse de la fonction htmlspecialchars.

*Exemple*

```
<?php
$texte = ’Olivier & Co. a déclaré : "c\’est l\’été !"’;
echo htmlspecialchars($texte,ENT_QUOTES);
?>
```

*Résultat dans le code source de la page (les éléments concernés sont en gras)*

```
Olivier &amp; Co. a déclaré : &quot;c&#039;est l&#039;été !&quot;
```

La fonction htmlentities présente un comportement identique à celui de htmlspecialchars mais pour tous les caractères ayant un équivalent en HTML (caractères accentués notamment).

*Syntaxe*

```
chaîne htmlentities(chaîne texte [, entier option [, chaîne
jeu [, booléen double_encodage]]])
```

`texte`

Chaîne à traiter.

`option`

Fonctionnement vis-à-vis des caractères guillemets (") et apostrophe (’).

`jeu`

Jeu de caractères à utiliser pour la conversion.

`double_encodage`

Indique s’il faut (`TRUE`, valeur par défaut) ou non (`FALSE`) encoder les entités HTML déjà encodées. Apparu en version 5.2.3.

Les deuxième et troisième paramètres ont la même signification que pour la fonction `htmlspecialchars`.

*Exemple*

```
<?php
$texte = ’Olivier & Co. a déclaré : "c\’est l\’été !"’;
echo htmlentities($texte,ENT_QUOTES);
?>
```

*Résultat dans le code source de la page (les éléments concernés sont en gras)*

```
Olivier &amp; Co. a d&eacute;clar&eacute; : &quot;c&#039;est
l&#039;&eacute;t&eacute; !&quot;
```

La fonction `nl2br` prend une chaîne de caractères et la retourne après avoir inséré une balise HMTL de saut de ligne devant chaque saut de ligne : la balise est `<br />` (compatibilité XHTML) à partir de la version 4.0.5 de PHP et `<br>` avant.

*Syntaxe*

```
chaîne nl2br(chaîne texte)
```

`texte`

Chaîne à traiter.

*Exemple*

```
<?php
$texte = "Première ligne.\nDeuxième ligne.";
echo nl2br($texte);
?>
```

*Résultat dans le code source de la page (les éléments concernés sont en gras)*

```
Première ligne.<br />
Deuxième ligne.
```

Enfin, la fonction `strip_tags` prend une chaîne de caractères et la retourne après avoir supprimé toutes les balises HTML qu’elle contient.

*Syntaxe*

```
chaîne strip_tags(chaîne texte[, chaîne balises_autorisées])
```

`texte`

Chaîne à traiter.

`balises_autorisées`

Liste des balises à conserver dans la chaîne.

Le deuxième paramètre permet d'indiquer les balises à conserver.

*Exemple*

```
<?php
$texte = "<b>Olivier</b> <i>Heurtel</i>";
echo $texte,'<br/>';
echo strip_tags($texte),'<br/>';
echo strip_tags($texte,'<b>'),'<br/>';
?>
```

*Résultat*

```
OlivierHeurtel
Olivier Heurtel
Olivier Heurtel
```

Ces fonctions vont donc permettre de gérer les différents problèmes évoqués précédemment.

Pour éviter tout problème d'affichage dans le navigateur, l'idée est d'appliquer une transformation aux données avant ou dans l'instruction echo.

Des fonctions personnelles peuvent être développées pour effectuer cette opération.

*Exemple*

```
<?php
// Fonction permettant d'afficher des données dans un formulaire.
// Encoder tous les caractères HTML spéciaux.
function  vers_formulaire($valeur) {
  return htmlentities($valeur,ENT_QUOTES);
}
// Fonction permettant d'afficher des données dans une page.
// Encoder tous les caractères HTML spéciaux.
// Transformer les sauts de lignes en <br />.
function  vers_page($valeur) {
  return nl2br(htmlentities($valeur,ENT_QUOTES));
}
?>
```

Ces fonctions peuvent ensuite être utilisées dans un script de traitement d'un formulaire.

*Exemple*

```
<?php
// Inclusion du fichier qui contient les définitions de nos
// fonctions générales.
include('fonctions.inc');
// Tester si la page est appelée après validation du formulaire
if (isset($_POST['ok'])) {
  // Appel de la fonction qui supprime l'encodage
  // "magic quotes" de tous les tableaux GPC.
  supprimer_encodage_MQ_GPC();
  // Récupération de la valeur saisie dans le formulaire
  $nom = isset($_POST['nom'])?$_POST['nom']:'';
  // La valeur saisie est réaffichée dans le formulaire et
  // dans la page ...
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
```

```
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom"
        value="<?php echo vers_formulaire($nom); ?>" />
      <input type="submit" name="ok" value="OK" /><br />
      <?php echo vers_page($nom); ?>
    </div>
    </form>
  </body>
</html>
```

*Résultat*

Nom : il dit : "<b>c'est l'été</b> OK
il dit : "<b>c'est l'été</b>"

Avec cette approche, la saisie est restituée telle quelle : si l'utilisateur a saisi une balise, il la retrouve (celle-ci n'est pas interprétée par le navigateur). Si besoin, les fonctions peuvent être modifiées pour supprimer les balises à l'aide de la fonction `strip_tags`.

---

La fonction `nl2br` doit être appelée après les fonctions `htmlentities` ou `htmlspecialchar`. Dans le cas contraire, la balise `<br />` ajoutée par `nl2br` est encodée (en `<br />`) avant d'être insérée dans le code source de la page et n'est donc pas interprétée par le navigateur (le texte `<br />` apparaît dans la page affichée). La fonction `nl2br` ne doit pas être appelée pour du texte destiné à une zone `<textarea>` (sous peine, là encore, d'avoir le texte `<br />` dans la zone).

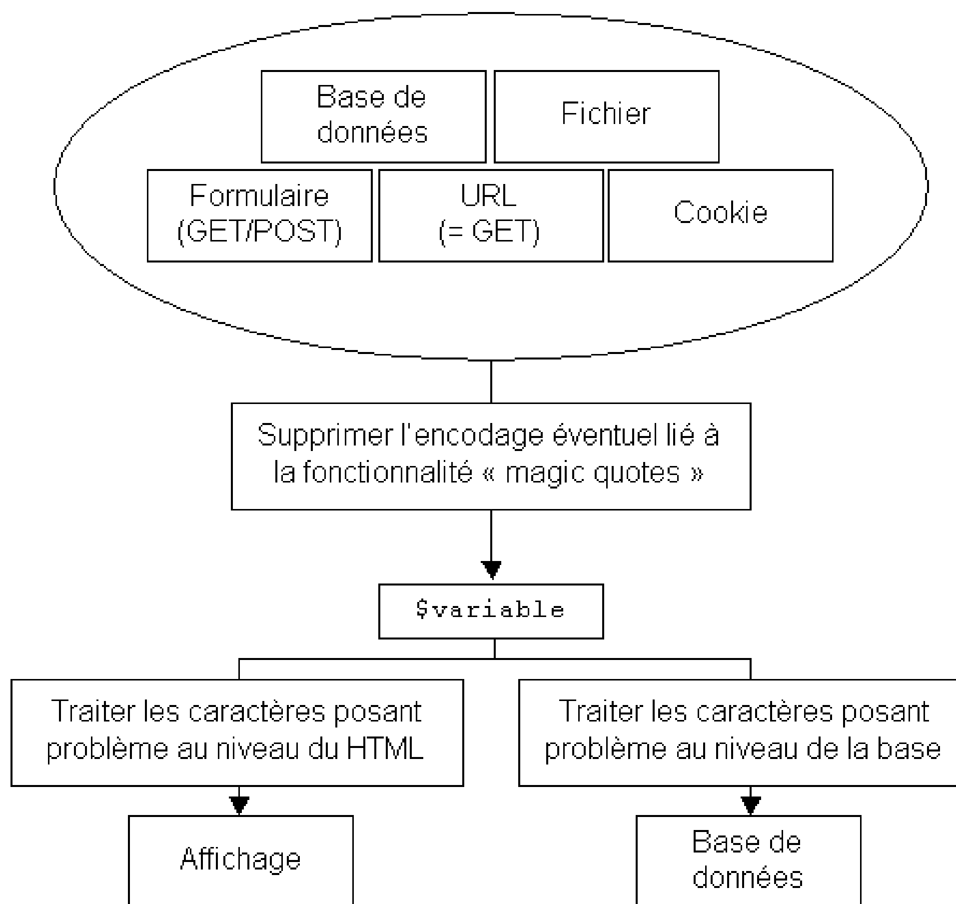
---

## 3. Synthèse

La gestion des formulaires et des URL, avec l'affichage et/ou l'enregistrement des données récupérées, peut rapidement devenir un casse-tête compte tenu des différents problèmes potentiels et des directives de configuration associées.

Néanmoins, avec un peu de rigueur et deux ou trois fonctions écrites une fois pour toute, cette gestion peut se faire sans soucis.

La stratégie préconisée est résumée par le schéma suivant :




Le cas des bases de données et des cookies est traité en détail dans les chapitres ultérieurs ; le cas des fichiers a été présenté dans le chapitre Utiliser les fonctions PHP.

Les fonctions `supprimer_encodage_MQ`, `supprimer_encodage_MQ_GPC`, `vers_formulaire`, et `vers_page` proposées en exemples dans ce chapitre permettent de gérer les données externes sans problème selon la stratégie préconisée ci-dessus.

# Utilisation des filtres

## 1. Principes

Depuis la version 5.2.0, une nouvelle extension permet de filtrer et valider des données, notamment celles provenant de la saisie des utilisateurs ou d'une URL.

Chaque filtre est défini par un numéro (identifiant), un nom et, éventuellement, des options et des indicateurs qui précisent le comportement du filtre. Chaque option est définie par un nom qui est utilisé comme clé dans un tableau associatif. Chaque indicateur est défini par une constante ; pour spécifier plusieurs indicateurs, il suffit de sommer les constantes correspondantes.

Quelques exemples de filtres (voir la documentation pour la description de tous les filtres) :

| **Identifiant (constante prédéfinie)** | **Description** |
|---|---|
| FILTER_VALIDATE_INT | Valide une valeur en tant qu'entier. Les options min_range et max_range permettent de définir un intervalle de validité. |
| FILTER_VALIDATE_FLOAT | Valide une valeur en tant que nombre à virgule flottante. |
| FILTER_VALIDATE_REGEXP | Valide une valeur à l'aide d'une expression régulière compatible PERL. Les expressions régulières PERL sont légèrement différentes des expressions régulières "PHP" présentées dans cet ouvrage (voir la documentation pour les différences). L'expression régulière à utiliser est spécifiée grâce à l'option regexp. |
| FILTER_VALIDATE_EMAIL | Valide une valeur en tant qu'adresse électronique. |
| FILTER_SANITIZE_STRING | Supprime les balises contenues dans une chaîne et encode les caractères ' et ". Plusieurs indicateurs sont disponibles pour supprimer ou encoder des caractères supplémentaires (voir ci-dessous). |
| FILTER_SANITIZE_SPECIAL_CHARS | Encode en HTML les caractères ', ", <, > et & ainsi que tous les caractères de code ASCII inférieur à 32. Plusieurs indicateurs sont disponibles pour supprimer ou encoder des caractères supplémentaires (voir ci-dessous). |
| FILTER_SANITIZE_MAGIC_QUOTES | Échappe les caractères ', " et \ (applique la fonction addslashes). |

Les indicateurs suivants peuvent être utilisés avec les filtres FILTER_SANITIZE_STRING et FILTER_SANITIZE_SPECIAL_CHARS :

FILTER_FLAG_STRIP_LOW

Supprime les caractères de code ASCII inférieur à 32.

FILTER_FLAG_STRIP_HIGH

Supprime les caractères de code ASCII supérieur à 127.

FILTER_FLAG_ENCODE_HIGH

Encode en HTML les caractères de code ASCII supérieur à 127.

En complément, les indicateurs suivants peuvent être utilisés avec le filtre FILTER_SANITIZE_STRING :

FILTER_FLAG_NO_ENCODE_QUOTES

N'encode pas les caractères ' et ".

FILTER_FLAG_ENCODE_AMP

Encode en HTML le caractère &.

FILTER_FLAG_ENCODE_LOW

Encode en HTML les caractères de code ASCII inférieur à 32.

De plus, l'indicateur FILTER_NULL_ON_FAILURE peut être utilisé avec tous les filtres pour que les fonctions retournent la valeur NULL au lieu de la valeur FALSE en cas d'échec.

Ces filtres peuvent être utilisés dans les fonctions filter_var, filter_var_array, filter_input et filter_input_array.

## Fonctions filter_var et filter_var_array

La fonction filter_var permet de filtrer une donnée.

*Syntaxe*

```
mixte filter_var(mixte donnée[,entier filtre[,mixte options_indicateurs]])
```

donnée

Donnée à filtrer.

filtre

Identifiant du filtre à appliquer (FILTER_SANITIZE_STRING par défaut).

options_indicateurs

Options et/ou indicateurs éventuels du filtre (voir ci-dessous).

Cette fonction retourne la donnée filtrée, ou FALSE si le filtre a échoué (ou NULL si l'indicateur FILTER_NULL_ON_FAILURE est utilisé).

Dans le cas le plus général, le paramètre options_indicateurs est spécifié sous la forme d'un tableau associatif comportant une ou deux lignes, avec les clés suivantes : flags pour les indicateurs et options pour les options. Comme indiqué précédemment, plusieurs indicateurs peuvent être fournis en sommant les constantes correspondantes. La valeur des options est elle aussi définie sous la forme d'un tableau associatif dans lequel le nom de l'option est utilisé comme clé.

Dans le cas où le paramètre options_indicateurs ne définit que des indicateurs, la valeur correspondante peut être passée directement, sans utiliser de tableau.

*Exemple 1*

```
<?php
echo "<b>Filtrer un nombre entier</b><br />";
$valeurs = array('123','abc','1.2',NULL);
foreach ($valeurs as $x) {
  echo "$x => ";
  var_export(filter_var($x,FILTER_VALIDATE_INT));
  echo "<br />";
}
echo "<b>+ NULL au lieu de FALSE en cas d'erreur</b><br />";
$x = 'abc';
// indicateur passé en option directement
$options = FILTER_NULL_ON_FAILURE;
echo "$x => ";
var_export(filter_var($x,FILTER_VALIDATE_INT,$options));
echo "<br />";
echo "<b>Filtrer un nombre entier (0-100)</b><br />";
// options du filtre définies via un tableau associatif
$options =
  array
```

```
    (
    'options' => array('min_range' => 0,'max_range' => 100)
    );
$valeurs = array('0','100','101');
foreach ($valeurs as $x) {
  echo "$x => ";
  var_export(filter_var($x,FILTER_VALIDATE_INT,$options));
  echo "<br />";
}
echo "<b>+ NULL au lieu de FALSE en cas d'erreur</b><br />";
// indicateur ajouté dans le tableau des options
$options =
  array
    (
    'options' => array('min_range' => 0,'max_range' => 100),
    'flags' => FILTER_NULL_ON_FAILURE
    );
$x = '101';
echo "$x => ";
var_export(filter_var($x,FILTER_VALIDATE_INT,$options));
echo "<br />";
echo "<b>Filtrer avec une expression régulière</b><br />";
$regexp = '<^[0-9]{2}/[0-9]{2}/[0-9]{4}$>';
$options =
  array
    (
    'options' => array('regexp' => $regexp)
    );
$valeurs = array('01/01/2007','01/01/07');
foreach ($valeurs as $x) {
  echo "$x => ";
  var_export(filter_var($x,FILTER_VALIDATE_REGEXP,$options));
  echo "<br />";
}
?>
```

*Résultat*

```
Filtrer un nombre entier
123 => 123
abc => false
1.2 => false
+ NULL au lieu de FALSE en cas d'erreur
abc => NULL
Filtrer un nombre entier (0-100)
0 => 0
100 => 100
101 => false
+ NULL au lieu de FALSE en cas d'erreur
101 => NULL
Filtrer avec une expression régulière
01/01/2007 => '01/01/2007'
01/01/07 => false
```

*Exemple 2*

```
<?php
$texte = "<b>c'est l'été</b>";
echo "// texte affiché sans précaution<br />\n";
echo $texte,"<br />\n";
echo "// FILTER_SANITIZE_SPECIAL_CHARS<br />\n";
echo
  filter_var
    (
    $texte,
    FILTER_SANITIZE_SPECIAL_CHARS
    ),
  "<br />\n";
echo "// + option FILTER_FLAG_ENCODE_HIGH<br />\n";
```

```
echo
  filter_var
    (
    $texte,
    FILTER_SANITIZE_SPECIAL_CHARS,
    FILTER_FLAG_ENCODE_HIGH
    ),
  "<br />\n";
echo "// FILTER_SANITIZE_STRING<br />\n";
echo filter_var($texte,FILTER_SANITIZE_STRING),"<br />\n";
echo "// + option FILTER_FLAG_ENCODE_HIGH<br />\n";
echo
  filter_var
    ($texte,
    FILTER_SANITIZE_STRING,
    FILTER_FLAG_ENCODE_HIGH
    ),
  "<br />\n";
echo "// FILTER_SANITIZE_MAGIC_QUOTES<br />\n";
echo
  filter_var
    (
    $texte,
    FILTER_SANITIZE_MAGIC_QUOTES
    ),
  "<br />\n";
?>
```

*Résultat dans le navigateur*

```
// texte affiché sans précaution
c'est l'été
// FILTER_SANITIZE_SPECIAL_CHARS
<b>c'est l'été</b>
// + option FILTER_FLAG_ENCODE_HIGH
<b>c'est l'été</b>
// FILTER_SANITIZE_STRING
c'est l'été
// + option FILTER_FLAG_ENCODE_HIGH
c'est l'été
// FILTER_SANITIZE_MAGIC_QUOTES
c\'est l\'été
```

*Résultat dans le source de la page*

```
// texte affiché sans précaution<br />
<b>c'est l'été</b><br />
// FILTER_SANITIZE_SPECIAL_CHARS<br />
&#60;b&#62;c&#39;est l&#39;été&#60;/b&#62;<br />
// + option FILTER_FLAG_ENCODE_HIGH<br />
&#60;b&#62;c&#39;est l&#39;&#233;t&#233;&#60;/b&#62;<br />
// FILTER_SANITIZE_STRING<br />
c&#39;est l&#39;été<br />
// + option FILTER_FLAG_ENCODE_HIGH<br />
c&#39;est l&#39;&#233;t&#233;<br />
// FILTER_SANITIZE_MAGIC_QUOTES<br />
<b>c\'est l\'été</b><br />
```

La fonction filter_var_array permet de filtrer un tableau de données.

*Syntaxe*

```
mixte filter_var_array(tableau données[,mixte filtres])
```

données

Tableau associatif contenant les données à filtrer. Les clés sont des chaînes de caractères qui identifient chaque donnée à valider.

filtres

Définition des filtres à appliquer à chaque donnée du tableau de données.

La fonction retourne le tableau des données filtrées ; les lignes pour lesquelles le filtre a échoué sont à FALSE (ou NULL si l'indicateur FILTER_NULL_ON_FAILURE est utilisé) et celles pour lesquelles la donnée n'existe pas sont à NULL.

Dans le cas le plus général, le paramètre filtres est spécifié sous la forme d'un tableau associatif qui reprend les clés du tableau de données (la correspondance entre les tableaux données et filtres s'effectue par l'intermédiaire de la clé). Chaque valeur du tableau filtres peut être un simple identifiant de filtre ou un tableau associatif donnant une description plus complète du filtre à appliquer. Dans ce cas, les clés du tableau filtres sont filter pour l'identifiant du filtre, flags pour les indicateurs à appliquer au filtre et options pour les options à appliquer au filtre ; les valeurs associées aux clés flags et options se définissent comme pour la fonction filter_var.

Dans le cas où le même filtre doit être appliqué à toutes les données, sans indicateur ni option, le paramètre filtres peut être un simple entier égal à l'identifiant du filtre.

_Exemple_

```
<?php
echo '<b>Filtrer un tableau de nombres entier</b><br />';
$valeurs = array('123','abc');
var_export($valeurs);
echo '<br /> => ';
// Même filtre à appliquer à toutes les données,
// sans indicateur ni option.
var_export
  (filter_var_array($valeurs,FILTER_VALIDATE_INT));
echo '<br />';
echo '<b>Filtrer un tableau de données diverses (1)</b><br />';
$valeurs = array
    (
    'age' => 123,
    'taille' => 'abc',
    'mail' => 'contact@olivier-heurtel.fr'
    );
// Filtre différent mais "simple" (sans indicateur
// ni option) à appliquer aux données.
$filtres = array
    (
    'age' => FILTER_VALIDATE_INT,
    'taille' => FILTER_VALIDATE_INT,
    'mail' => FILTER_VALIDATE_MAIL
    );
var_export($valeurs);
echo '<br /> => ';
var_export(filter_var_array($valeurs,$filtres));
echo '<br />';
echo '<b>Filtrer un tableau de données diverses (2)</b><br />';
$valeurs = array
    (
    'age' => 123,
    'taille' => 'abc',
    'mail' => 'contact@olivier-heurtel.fr'
    );
// Filtre avec options et indicateur à appliquer à une
// des données.
$filtre_age = array
    (
    'filter' => FILTER_VALIDATE_INT,
    'options' => array('min_range' => 0,'max_range' => 100),
    'flags' => FILTER_NULL_ON_FAILURE
    );
// Noter la mention d'un filtre pour une donnée
// qui n'existe pas.
$filtres = array
    (
    'age' => $filtre_age,
    'taille' => FILTER_VALIDATE_INT,
    'poids' => FILTER_VALIDATE_INT, // n'existe pas
```

```
    'mail' => FILTER_VALIDATE_MAIL
    );
var_export($valeurs);
echo '<br />';
var_export(filter_var_array($valeurs,$filtres));
?>
```

*Résultat*

```
Filtrer un tableau de nombres entier
array ( 0 => '123', 1 => 'abc', )
=> array ( 0 => 123, 1 => false, )
Filtrer un tableau de données diverses (1)
array ( 'age' => 123, 'taille' => 'abc', 'mail' =>
'contact@olivier-heurtel.fr', )
=> array ( 'age' => 123, 'taille' => false, 'mail' =>
'contact@olivier-heurtel.fr', )
Filtrer un tableau de données diverses (2)
array ( 'age' => 123, 'taille' => 'abc', 'mail' =>
'contact@olivier-heurtel.fr', )
=> array ( 'age' => NULL, 'taille' => false, 'poids' => NULL, 'mail' =>
'contact@olivier-heurtel.fr', )
```

Dans le dernier exemple, notez la valeur NULL qui a été associée à la donnée poids (mentionnée dans le filtre mais absente des données).

### Fonctions filter_input et filter_input_array

Les fonctions filter_input et filter_input_array sont similaires aux fonctions filter_var et filter_var_array mais s'appliquent à des données extérieures à PHP (données d'un formulaire par exemple) et non à des variables du script.

*Syntaxe*

```
mixte filter_input(entier source,chaîne nom_variable[,entier
filtre[,mixte options_indicateurs]])
```

source

Source des données. Une des constantes INPUT_GET (données passées par la méthode GET), INPUT_POST (données passées par la méthode POST), INPUT_COOKIE (données passées par un cookie).

nom_variable

Nom de la variable à traiter.

filtre

Identifiant du filtre à appliquer (FILTER_DEFAULT par défaut).

options_indicateurs

Options et/ou indicateurs éventuels du filtre (identique à la fonction filter_var).

Cette fonction retourne la donnée filtrée, FALSE si le filtre a échoué ou NULL si la variable n'est pas définie. Si l'indicateur FILTER_NULL_ON_FAILURE est utilisé, la fonction retourne NULL si le filtre a échoué ou FALSE si la variable n'est pas définie.

*Syntaxe*

```
mixte filter_input_array(entier source[,mixte filtres])
```

source

Source des données (identique à la fonction filter_input).

filtres

Définition des filtres à appliquer à chaque donnée du tableau de données (identique à la fonction filter_var_input).

Cette fonction est équivalente à un appel à la fonction filter_var_array effectué sur le tableau $_GET, $_POST ou $_COOKIE correspondant à la source (INPUT_GET, INPUT_POST ou INPUT_COOKIE).

La fonction retourne le tableau des données filtrées ou NULL si la source ne contient aucune donnée ; les lignes pour lesquelles le filtre a échoué sont à FALSE (ou NULL si l'indicateur FILTER_NULL_ON_FAILURE est utilisé) et celles pour lesquelles la variable n'existe pas sont à NULL.

Un exemple d'utilisation de ces fonctions est donné dans la suite (cf. Application aux formulaires).

## 2. Application aux formulaires

Les filtres peuvent être utilisés pour implémenter tout ou partie des traitements relatifs à la gestion des formulaires :

- récupération des données saisies (fonctions filter_input et filter_input_array) ;
- vérification des données saisies (filtres FILTER_VALIDATE_*) ;
- traitement des problèmes relatifs aux données saisies (filtres FILTER_SANITIZE_*).

Il faut noter notamment que les filtres FILTER_SANITIZE_* effectuent des transformations similaires à celles des fonctions htmlspecialchars, html_entities et strip_tags, présentées précédemment (cf. E - Les problèmes sur les données saisies), et peuvent donc les remplacer dans nos exemples.

*Exemple*

```
<?php
// Inclusion du fichier qui contient les définitions de nos
// fonctions générales.
include('fonctions.inc');
// Tester si la page est appelée après validation du formulaire
if (filter_has_var(INPUT_POST, 'ok')) {
  // Définir les filtres pour les données saisies.
  $filtres =
    array
      (
      'nom' => array('filter'=> FILTER_SANITIZE_STRING,
                     'flags' => FILTER_FLAG_ENCODE_HIGH +
                                FILTER_FLAG_ENCODE_LOW)
      );
  // Récupérer la saisie filtrée.
  $saisie = filter_input_array(INPUT_POST,$filtres);
  $nom = $saisie['nom'];
  // La valeur saisie est réaffichée dans le formulaire et
  // dans la page ...
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom"
        value="<?php echo $nom; ?>" />
      <input type="submit" name="ok" value="OK" /><br />
      <?php echo $nom; ?>
    </div>
    </form>
  </body>
</html>
```

*Saisie*

Nom : il dit : "<b>c'est l'été</b> OK

*Résultat*

Nom : il dit : "c'est l'été" OK

il dit : "c'est l'été"

Sur cet exemple, le filtre `FILTER_SANITIZE_STRING` est utilisé pour récupérer les données saisies, supprimer les éventuelles balises et encoder les caractères pouvant poser un problème en cas de réaffichage (dans la page ou dans le formulaire).

# Aller sur une autre page

Dans le traitement effectué par un script PHP, il peut être nécessaire d’afficher une autre page.

Le cas peut se produire par exemple, à la fin du traitement du formulaire, la situation pouvant varier selon que le formulaire est traité par le script qui l’affiche ou par un script indépendant.

*Variantes possibles :*

| | | |
|---|---|---|
| OK | • Aller sur une autre page | • Page déjà bonne<br>• Aller sur une autre page |
| Problème | • Réafficher le formulaire avec un message<br>• Aller sur une page d’erreur spécifique | • Réafficher le formulaire avec un message<br>• Aller sur une page d’erreur spécifique<br>• Afficher l’erreur dans la page courante |

Rediriger l’utilisateur vers une autre page en cours de script est possible en utilisant la fonction `header` qui permet d’envoyer des en-têtes HTTP avec la page HTML (cf. Utiliser les fonctions PHP - Manipuler les en-têtes HTTP).

Nous allons utiliser l’en-tête `location` qui redirige la requête vers une autre adresse.

*Syntaxe de la directive `location`*

```
location: URL absolue ou relative
```

*Syntaxe avec la fonction header*

```
header(’location: URL absolue ou relative’)
```

*Exemples*

```
// Redirection vers un script PHP situé au même niveau.
header(’location: erreur.php’);
// Redirection vers une page HTML située à un sous-niveau.
header(’location: ./erreur/message.htm’);
// Redirection vers un autre site.
header(’location: http://www.olivier-heurtel.fr’);
```

Le protocole HTTP 1.1 requiert une URL absolue dans la directive `location`. Pour cela, vous pouvez utiliser les variables globales `$_SERVER[’HTTP_HOST’]` et `$_SERVER[’PHP_SELF’]` (cf. chapitre Annexes - Variables PHP prédéfinies).

*Exemple*

```
<?php
$url_relative = ’erreur.php’;
echo ’$url_relative = ’,$url_relative,’<br />’;
echo ’$_SERVER[\’HTTP_HOST\’] = ’,
      $_SERVER[’HTTP_HOST’],’<br />’;
echo ’$_SERVER[\’PHP_SELF\’] = ’,
      $_SERVER[’PHP_SELF’],’<br />’;
echo ’dirname($_SERVER[\’PHP_SELF\’]) = ’,
      dirname($_SERVER[’PHP_SELF’]),’<br />’;
$url_absolue = ’http://’ . $_SERVER[’HTTP_HOST’] .
               rtrim(dirname($_SERVER[’PHP_SELF’]), ’/\\’) .
               ’/’ . $url_relative;
echo ’$url_absolue = ’,$url_absolue,’<br />’;
?>
```

*Exemple*

```
$url_relative = erreur.php
$_SERVER['HTTP_HOST'] = xampp
$_SERVER['PHP_SELF'] = /eni/index.php
dirname($_SERVER['PHP_SELF']) = /eni
$url_absolue = http://xampp/eni/erreur.php
```

*Exemple simple d'utilisation sur un script `info.php`*

```
<?php
// Affecter une valeur à $nom si un nombre tiré aléatoirement
// entre 0 et 1 est égal à 1.
$nom = (rand(0,1)==1)?'Olivier':'';
// Tester si $nom est renseigné.
if ($nom == '') {
  // La variable $nom est vide, ce n'est pas normal :
  // => rediriger l'utilisateur vers une page d'erreur.
  header('location: erreur.htm');
  // Interrompre l'exécution de ce script.
  exit;
}
// La variable $nom n'est pas vide, laisser le script se poursuivre.
$message = "Bonjour $nom !"; // préparer un message
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
  <p><?php echo $message; ?></p>
  </body>
</html>
```

*Page `erreur.htm`*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Erreur</title></head>
  <body>
  <div>
  Le site est actuellement indisponible ; essayez plus tard.<br />
  Merci de votre compréhension.<br />
  <!-- Lien pour une nouvelle tentative -->
  <a href="info.php">Essayer de nouveau</a>
  </div>
  </body>
</html>
```

Statistiquement, une fois sur deux, l'appel du script `info.php` donnera le résultat suivant :

```
Bonjour Olivier !
```

Et donc, une fois sur deux, le suivant :

```
Le site est actuellement indisponible ; essayez plus tard.
Merci de votre compréhension.
Essayer de nouveau
```



Sauf cas particulier, la fonction `header` doit être appelée avant toute instruction (PHP ou HTML) qui a pour effet de commencer à construire la page HTML (cf chapitre Utiliser les fonctions PHP - Manipuler les en-têtes HTTP).

## Premier exemple de logique d'enchaînement

Un script saisie.php assure l'affichage initial du formulaire et son traitement. En cas d'erreur, le formulaire est proposé de nouveau pour correction (avec les valeurs saisies), accompagné d'un message d'erreur ; en cas de succès du traitement, une autre page est appelée.

```
<?php
// Inclure le fichier qui contient les définitions de nos
// fonctions générales.
include('fonctions.inc');
// Tester comment le script est appelé.
if (isset($_POST['ok'])) {
  // Traitement du formulaire.
  // Appeller la fonction qui supprime l'encodage
  // "magic quotes" de tous les tableaux GPC.
  supprimer_encodage_MQ_GPC();
  // Récupérer les valeurs saisies dans le formulaire.
  $nom = trim($_POST['nom']);
  // Contrôler les valeurs saisies.
  if ($nom == '')
    { $message .= "Le nom est obligatoire.\n"; }
  if (strlen($nom) > 10)
    { $message .= "Le nom doit avoir au plus 10 caractères.\n"; }
  // Tester s'il y a des erreurs.
  if ($message == '') {
    // Pas d'erreur.
    // Rediriger l'utilisateur vers une autre page et interrompre
    // l'exécution du script.
    header('location: accueil.php');
    exit;
  } else {
    // Erreur.
    // Préparer le message pour l'affichage.
    $message = vers_page($message);
  }
} else {
  // Affichage initial.
  // Sur cet exemple simple, rien à faire.
}
// Dans le code HTML qui suit, inclusion de deux petits bouts de
// code PHP pour afficher respectivement la valeur des zones de
// saisie et le message.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisie</title></head>
  <body>
    <form action="saisie.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom"
        value="<?php echo vers_formulaire($nom); ?>" />
      <input type="submit" name="ok" value="OK" /><br />
      <?php echo $message; ?>
    </div>
    </form>
  </body>
</html>
```

## Deuxième exemple de logique d'enchaînement

Un script saisie.htm assure l'affichage initial du formulaire et un script traitement.php le traitement. En cas d'erreur, un message d'erreur est affiché et l'utilisateur est invité à revenir en arrière. En cas de succès du traitement, une autre page est affichée.

*Fichier saisie.htm*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
  <head><title>Saisie</title></head>
  <body>
    <form action="traitement.php" method="post">
    <div>
      Nom :
      <input type="text" name="nom" value="" />
      <input type="submit" name="ok" value="OK" />
    </div>
    </form>
  </body>
</html>
```

*Script traitement.php*

```
<?php
// Inclure le fichier qui contient les définitions de nos
// fonctions générales.
include('fonctions.inc');
// Tester comment le script est appelé.
if (isset($_POST['ok'])) {
  // Traitement du formulaire.
  // Appeller la fonction qui supprime l'encodage
  // "magic quotes" de tous les tableaux GPC.
  supprimer_encodage_MQ_GPC();
  // Récupérer les valeurs saisies dans le formulaire.
  $nom = trim($_POST['nom']);
  // Contrôler les valeurs saisies.
  if ($nom == '')
    { $message .= "Le nom est obligatoire.\n"; }
  if (strlen($nom) > 10)
    { $message .= "Le nom doit avoir au plus 10 caractères.\n"; }
  // Tester s'il y a des erreurs.
  if ($message == '') {
    // Pas d'erreur.
    // Rediriger l'utilisateur vers une autre page et interrompre
    // l'exécution du script.
    header('location: accueil.php');
    exit;
  } else {
    // Erreur.
    // Préparer le message pour l'affichage.
    $message = vers_page($message);
  }
}
// Dans le code HTML qui suit, inclusion d'un petit bout de
// code PHP pour afficher le message.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Erreur</title></head>
  <body>
    <!-- Petit formulaire contenant un bouton permettant
    ---- de revenir en arrière (avec du JavaScript) pour corriger.
    -->
    <form>
    <div>
      <?php echo $message; ?><br />
      <input type="button" value="Corriger"
                      onClick="self.history.back()">
    </div>
    </form>
  </body>
</html>
```

D'autres logiques d'enchaînement peuvent exister, la fonction `header` permettant d'envisager différents cas de figure (voir la documentation PHP).

Le résultat du traitement peut aussi être affiché dans une autre fenêtre grâce à l'attribut `target` de la balise `<form>` :

```
<form action="traitement.php" method="post" target="traitement">
```

Si la fenêtre n’existe pas, elle sera créée par le navigateur.

# Échanger un fichier entre le client et le serveur

## 1. Vue d'ensemble

Certains sites peuvent proposer aux utilisateurs de transférer des documents de leur poste vers le serveur Web : déposer un CV sur un site (site de recherche d'emploi), mettre une pièce-jointe dans un message (site de messagerie) ou simplement stocker le document sur le serveur (site de stockage).

Dans la terminologie anglo-saxonne, cette fonctionnalité s'appelle le "file upload".

Inversement, beaucoup de sites permettent aux utilisateurs de télécharger ("download") des documents du serveur Web vers leur poste.

Ces deux fonctionnalités sont des applications particulières des techniques présentées dans ce chapitre.

## 2. Télécharger un fichier à partir du client : "file upload"

Cette fonctionnalité, très simple à mettre en œuvre en PHP, nécessite deux opérations :

- dans un formulaire, proposer une zone permettant à l'utilisateur de désigner l'emplacement du fichier sur son poste ;
- dans le script de traitement du formulaire, récupérer le fichier envoyé par l'utilisateur et en faire quelque chose.

Dans la première partie de ce chapitre, nous avions vu la possibilité de mettre dans un formulaire une zone permettant d'indiquer l'emplacement d'un fichier sur son poste (`type="file"`).

Mettre une zone de ce type n'est pas suffisant. Pour provoquer le transfert du fichier, il faut ajouter l'attribut `enctype="multipart/form-data"` dans la balise `<form>` :

```
<form action="saisie.php" method="post"
                   enctype="multipart/form-data">
```

 Cette technique ne fonctionne qu'avec les formulaires qui utilisent la méthode `POST`.

En complément, il est possible d'ajouter une zone cachée dans le formulaire afin de limiter la taille des fichiers qui peuvent être envoyés vers le serveur. Cette zone cachée, obligatoirement située <u>avant</u> la zone de type `file` doit s'appeler `MAX_FILE_SIZE` (attribut `name`) et préciser la taille maximum en octets dans l'attribut `value` :

<u>*Exemple de zone cachée pour limiter la taille des fichiers à 10 ko :*</u>

```
<input type="hidden" name="MAX_FILE_SIZE" value="10240">
```

La valeur précisée dans cette zone ne peut pas être supérieure à la valeur de la directive de configuration `upload_max_filesize` (2 Mo par défaut). Si la zone cachée n'est pas présente, c'est la taille spécifiée dans la directive `upload_max_filesize` qui s'applique.

Par ailleurs, le téléchargement n'est possible que si la directive de configuration `file_uploads` est à `on`.

<u>*Exemple de formulaire complet*</u>

```
<form action="saisie.php" method="post"
                 enctype="multipart/form-data">
<div>
  Fichier :
  <input type="file" name="fichier" value="" />
  <input type="submit" name="ok" value="OK" />
</div>
</form>
```

Lorsqu'un fichier est envoyé avec un formulaire, des informations sur ce fichier sont accessibles dans le script PHP grâce à la variable `$_FILES` ; la valeur saisie par l'utilisateur n'est plus disponible dans `$_POST`.

$_FILES est un tableau associatif multidimensionnel ; la première clé est égale au nom de la zone de type file du formulaire (fichier sur notre exemple) et le tableau associatif associé présente cinq lignes :

| Clé | Valeur |
|---|---|
| name | Nom du fichier (sans chemin d'accès) |
| type | Type MIME du fichier (fourni par le navigateur) |
| size | Taille du fichier en octets |
| tmp_name | Nom du fichier temporaire créé sur le serveur (chemin complet) |
| error | Code d'erreur. Une des constantes suivantes :<br>UPLOAD_ERR_OK (0) : pas d'erreur<br>UPLOAD_ERR_INI_SIZE (1) : taille du fichier supérieure à la taille définie par la directive de configuration upload_max_filesize<br>UPLOAD_ERR_FORM_SIZE (2) : taille du fichier supérieure à la taille définie par l'option MAX_FILE_SIZE du formulaire<br>UPLOAD_ERR_PARTIAL (3) : le fichier a été chargé partiellement<br>UPLOAD_ERR_NO_FILE (4) : aucun fichier saisi<br>UPLOAD_ERR_NO_TMP_DIR (6) : pas de répertoire temporaire. Introduit en version 5.0.3.<br>UPLOAD_ERR_CANT_WRITE (7) : erreur lors de l'écriture du fichier sur disque. Introduit en version 5.1.0.<br>UPLOAD_ERR_EXTENSION (8) : transfert stoppé par l'extension. Introduit en version 5.2.0. |

Selon les versions, il existe un code erreur 5, sans nom de constante, en cas d'erreur dans le fichier saisi (nom, chemin). Ce code d'erreur n'existe plus depuis la version 5.2.

Sur le serveur, le fichier transféré est un fichier temporaire portant un nom du type php*.tmp, situé dans le répertoire défini par la directive de configuration upload_tmp_dir.

La structure de $_FILES permet d'avoir plusieurs zones de type file dans le formulaire et donc d'autoriser le chargement de plusieurs fichiers.

*Exemple :*

*Le formulaire comporte deux zones de type file nommées fichier1 et fichier2) :*

| | | |
|---|---|---|
| fichier1 | name | Photo.gif |
| | type | image/gif |
| | tmp_name | d:\temp\phpB.tmp |
| | size | 2376 |
| fichier2 | name | cv.pdf |
| | type | application/pdf |
| | tmp_name | d:\temp\php12.tmp |
| | size | 52147 |

Si le fichier temporaire créé sur le serveur n'est pas exploité (renommé/copié/déplacé) par le script PHP qui traite le formulaire, il est supprimé automatiquement à la fin du script. Dans le script PHP, il convient donc de manipuler le fichier temporaire selon les besoins de l'application.

*Exemple complet*

```php
<?php
// Inclusion du fichier qui contient les fonctions générales.
include('fonctions.inc');
// Initialisation de la variable de message.
$message = '';
// Traitement du formulaire.
if (isset($_POST['ok'])) {
  // Récupérer les informations sur le fichier.
  $informations = $_FILES["fichier"];
  // En extraire :
  //    - son nom
  $nom = valeur_saisie($informations['name']);
  //    - son type MIME
  $type_mime = $informations['type'];
  //    - sa taille
  $taille = $informations['size'];
  //    - l'emplacement du fichier temporaire
  $fichier_temporaire = $informations['tmp_name'];
  //    - le code d'erreur
  $code_erreur = $informations['error'];
  // Contrôles et traitement
  switch ($code_erreur) {
  case UPLOAD_ERR_OK :
    // Fichier bien reçu.
    // Déterminer sa destination finale
    $destination = "/app/documents/$nom";
    // Copier le fichier temporaire (tester le résultat).
    if (copy($fichier_temporaire,$destination)) {
      // Copie OK => mettre un message de confirmation.
      $message  = "Transfert terminé - Fichier = $nom - ";
      $message .= "Taille = $taille octets - ";
      $message .= "Type MIME = $type_mime.";
    } else {
      // Problème de copie => mettre un message d'erreur.
      $message = 'Problème de copie sur le serveur.';
    }
    break;
  case UPLOAD_ERR_NO_FILE :
    // Pas de fichier saisi.
    $message = 'Pas de fichier saisi.';
    break;
  case UPLOAD_ERR_INI_SIZE :
    // Taille fichier > upload_max_filesize.
    $message  = "Fichier '$nom' non transféré ";
    $message .= ' (taille > upload_max_filesize).';
    break;
  case UPLOAD_ERR_FORM_SIZE :
    // Taille fichier > MAX_FILE_SIZE.
    $message  = "Fichier '$nom' non transféré ";
    $message .= ' (taille > MAX_FILE_SIZE).';
    break;
  case UPLOAD_ERR_PARTIAL :
    // Fichier partiellement transféré.
    $message  = "Fichier '$nom' non transféré ";
    $message .= ' (problème lors du tranfert).';
    break;
  case UPLOAD_ERR_NO_TMP_DIR :
    // Pas de répertoire temporaire.
    $message  = "Fichier '$nom' non transféré ";
    $message .= ' (pas de répertoire temporaire).';
    break;
  case UPLOAD_ERR_CANT_WRITE :
    // Erreur lors de l'écriture du fichier sur disque.
    $message  = "Fichier '$nom' non transféré ";
    $message .= ' (erreur lors de l\'écriture du fichier sur disque).';
    break;
  case UPLOAD_ERR_EXTENSION :
    // Transfert stoppé par l'extension.
    $message  = "Fichier '$nom' non transféré ";
```

```
      $message .= ' (transfert stoppé par l\'extension).';
      break;
    default :
      // Erreur non prévue !
      $message  = "Fichier non transféré ";
      $message .= " (erreur inconnue : $code_erreur ).";
    }
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Upload</title></head>
  <body>
    <form action="upload.php" method="post"
                  enctype="multipart/form-data">
    <div>
      Fichier :
      <input size="100" type="file" name="fichier" value="" />
      <input type="submit" name="ok" value="OK" /><br />
      <?php echo vers_page($message); ?>
    </div>
    </form>
  </body>
</html>
```

*Résultat*

- Affichage initial du formulaire et sélection d'un fichier (à l'aide du bouton **Parcourir**...) :

Fichier : O:\Temp\photo.jpg Parcourir... OK

- Résultat du clic sur le bouton **OK** :

Fichier : Parcourir... OK
Transfert terminé - Fichier = photo.jpg - Taille = 165697 octets - Type MIME = image/jpeg.

Sur cet exemple, le fichier est copié dans un répertoire sur le serveur. Dans le chapitre Accéder à une base de données MySQL, nous verrons comment stocker le fichier dans une base de données MySQL.

## 3. Télécharger un fichier à partir du serveur : "download"

Pour télécharger un fichier à partir du serveur, il est possible d'utiliser la méthode classique qui consiste à utiliser un lien (balise <a>).

*Exemple*

```
<a href="cv.pdf">Télécharger</a>
```

Cette technique peut poser un certain nombre de problèmes :

- C'est le navigateur qui décide de proposer à l'utilisateur un dialogue d'enregistrement ou d'afficher directement le document s'il sait comment faire ;

- Les fichiers qui sont interprétés par le serveur (`.php` par exemple) ou par le navigateur (`.htm` par exemple) ne peuvent pas être chargés de cette manière.

Une autre technique est utilisable si vous souhaitez forcer le navigateur à proposer à l'utilisateur un dialogue d'enregistrement, et ce, pour n'importe quel type de document.

Cette technique consiste à envoyer certains en-têtes particuliers à l'aide de la fonction `header` (cf. Utiliser les fonctions PHP - Manipuler les en-têtes HTTP), suivi du document proprement dit.

Il n'est pas garanti que cette technique fonctionne correctement, ou de la même manière, avec tous les navigateurs. Il y a notamment des problèmes connus avec certaines versions d'Internet Explorer.

Les en-têtes minimums à envoyer sont les suivants :

```
Content-Disposition: attachment; filename=...
```

Cet en-tête suggère au navigateur de traiter le document comme une pièce jointe (`attachment`) et de proposer à l'utilisateur de l'enregistrer sous le nom défini par `filename`.

```
Content-Type: ...
```

Cet en-tête communique au navigateur le type MIME du document.

Quelques types MIME courants :

| **Type MIME** | **Nature du document** |
|---|---|
| `application/msword` | Document Microsoft Word |
| `application/octetstream` | Générique |
| `application/pdf` | Document PDF |
| `application/vnd.ms-excel` | Document Microsoft Excel |
| `application/vnd.ms-powerpoint` | Document Microsoft PowerPoint |
| `application/zip` | Document Zip |
| `image/bmp` | Image au format bitmap |
| `image/gif` | Image au format GIF |
| `image/jpeg` | Image au format JPEG |
| `image/tiff` | Image au format TIFF |
| `image/png` | Image au format PNG |
| `text/html` | Document HMTL |
| `text/plain` | Document texte |

En mettant n'importe quoi comme type MIME (`x/y` par exemple), le navigateur doit normalement se débrouiller avec l'extension du document.

Après l'envoi des en-têtes, il ne reste plus qu'à envoyer le document "directement" dans la page, par exemple à l'aide de la fonction `readfile` (cf. chapitre Utiliser les fonctions PHP - Manipuler les fichiers sur le serveur).

Cette technique va être illustrée à l'aide d'un script `download.php` qui propose une liste de documents en téléchargement. Nous présenterons deux exemples : un exemple qui utilise un formulaire et un exemple qui utilise des liens.

Dans les deux cas, la liste des documents qui peuvent être téléchargés est codée en dur dans le script. Dans une vraie application, cette liste de document viendrait sans doute d'une base de données (cf. chapitre Accéder à une base de données MySQL).

## Utilisation d'un formulaire

Dans ce premier exemple, le script génère un formulaire qui comporte un bouton de type "image" (`type="image"`) pour chaque fichier proposé en téléchargement, le nom du bouton étant égal au numéro du document.

Comme nous l'avons vu dans le chapitre Utiliser les fonctions MySQL, nous récupérerons dans `$_POST` deux variables

n_x et n_y donnant respectivement la position relative horizontale et verticale du clic à l'intérieur de l'image (n étant le nom de l'image cliquée, dans notre cas un numéro).

*Exemple*

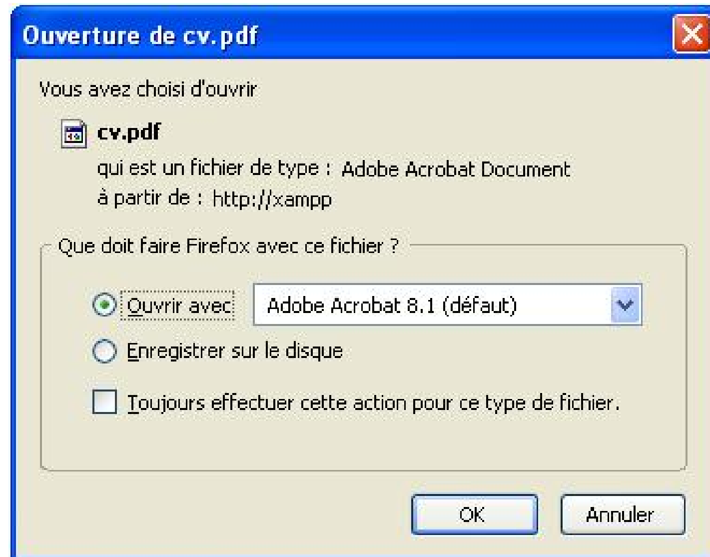
```
<?php
// Liste des documents (viendrait sans doute d'une
// base de données dans une vraie application).
$documents = array('cv.pdf','photo.gif');
// Traitement du formulaire si $_POST non vide
if (! empty($_POST)) {
  // Récupérer le numéro du document.
  // Prendre la clé de la première ligne de $_POST
  // (normalement du type n_x, n étant le numéro du document).
  list($numéro) = each($_POST);
  // Convertir la chaîne en entier => seul le n° reste.
  $numéro = (integer) $numéro;
  // En déduire le nom du document.
  $nom_fichier = $documents[$numéro];
  // Envoyer l'en-tête d'attachement.
  $header  = "Content-Disposition: attachment; ";
  $header .= "filename=$nom_fichier\n" ;
  header($header);
  // Envoyer l'en-tête du type MIME (ici, "inconnu").
  header("Content-Type: x/y\n");
  // Envoyer le document.
  // Pas d'encodage magic quotes avant de lire le fichier.
  set_magic_quotes_runtime(0);
  readfile($nom_fichier);
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Download</title></head>
  <body>
    <form action="Download.php" method="post">
    <table border="1" cellpadding="4" cellspacing="0">
    <tr align="center">
      <th>document</th><th>télécharger</th>
    </tr>
    <?php
    // Un petit bout de code PHP pour générer les lignes du
    // tableau présentant la liste des documents.
    // Parcourir la liste des documents et utiliser le nom
    // pour l'affichage et le numéro comme nom de l'image.
    foreach($documents as $numéro => $document) {
      echo sprintf
          (
          "<tr><td>%s</td><td align=\"center\">%s</td></tr>\n",
          $document,
          "<input type=\"image\" name=\"$numéro\"
          src=\"download.gif\" />"
          );
    }
    ?>
    </table>
    </form>
  </body>
</html>
```

*Résultat (dans Firefox)*

- Affichage initial de la page

| document | télécharger |
|---|---|
| cv.pdf | |
| photo.gif | |

- Après clic sur l’image associée au document `cv.pdf`




## Utilisation de liens

Dans ce deuxième exemple, le script génère un tableau qui comporte un lien pour chaque fichier proposé en téléchargement. Le lien appelle de nouveau le script en passant le numéro du document en paramètre dans l’URL.

*Exemple*

```
<?php
// Liste des documents (viendrait sans doute d’une
// base de données dans une vraie application).
$documents = array(’cv.pdf’,’photo.gif’);
// Traitement du formulaire si $_GET non vide
if (! empty($_GET)) {
  // Récupérer le numéro du document.
  $numéro = $_GET[’no’];
  // En déduire le nom du document.
  $nom_fichier = $documents[$numéro];
  // Envoyer l’en-tête d’attachement.
  $header  = "Content-Disposition: attachment; ";
  $header .= "filename=$nom_fichier\n" ;
  header($header);
  // Envoyer l’en-tête du type MIME (ici, "inconnu").
  header("Content-Type: x/y\n");
  // Envoyer le document.
  // Pas d’encodage magic quotes avant de lire le fichier.
  set_magic_quotes_runtime(0);
  readfile($nom_fichier);
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Download</title></head>
  <body>
    <table border="1" cellpadding="4" cellspacing="0">
    <tr align="center"><th>document</th></tr>
    <?php
```

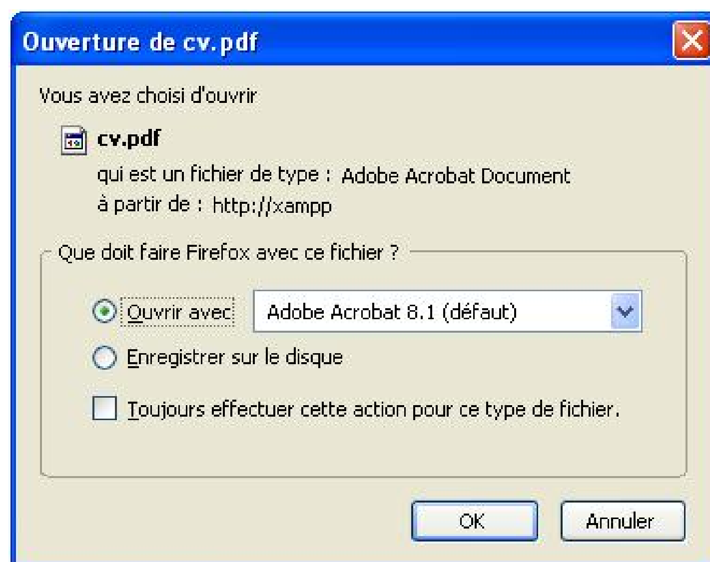
```
    // Un petit bout de code PHP pour générer les lignes du
    // tableau présentant la liste des documents.
    // Parcourir la liste des documents et utiliser le nom
    // pour l'affichage et le numéro dans.
    foreach($documents as $numéro => $document) {
      echo sprintf
          (
          "<tr><td>%s</td></tr>\n",
          "<a href=\"download.php?no=$numéro\">$document</a>"
          );
    }
    ?>
    </table>
  </body>
</html>
```

*Résultat (dans Firefox)*

- Affichage initial de la page

| document |
|---|
| cv.pdf |
| photo.gif |

- Après clic sur le lien associé au document `cv.pdf`



# Introduction

## 1. Vue d’ensemble

L’utilisation d’une base de données SQL est souvent indispensable pour mettre en place un site Web dynamique. C’est en effet un moyen standardisé de stocker des données utiles pour le site :

- liste des utilisateurs avec leurs préférences ;
- catalogue de produits ;
- trace des transactions effectuées ...

PHP propose un support natif pour un grand nombre de bases de données, parmi lesquelles MySQL, Oracle, Microsoft SQL Server, Informix, Sybase, SQLite. Par ailleurs, PHP supporte ODBC (*Open DataBase Connectivity*) et peut donc accéder à toute base de données supportant ODBC.

Dans cet ouvrage, nous étudierons l’accès à une base de données MySQL.

Typiquement, lors de l’utilisation d’une base de données, le script PHP aura besoin d’effectuer une ou plusieurs des tâches suivantes :

- se connecter et se déconnecter ;
- lire des données (une ligne ou plusieurs lignes) ;
- mettre à jour des données (ajout, modification ou suppression).

Les différentes tâches types seront étudiées dans ce chapitre.

Pour les différents exemples, nous utiliserons la base de données `eni` créée dans le chapitre Introduction à MySQL. Pour obtenir les mêmes résultats que ceux présentés dans ce chapitre, vous devez récréer la base de données.

*Exemple*

```
[root@xampp ~]# mysql -u root < creer-base-eni.sql
```

## 2. Quelle extension utiliser pour accéder à MySQL ?

Depuis la version 5, PHP propose deux extensions pour accéder à une base de données MySQL :

- MySQL (préfixe `mysql_`) ;
- MySQLi (préfixe `mysqli_`).

L’extension MySQL est l’ancienne extension, présente dans les versions antérieures de PHP. Cette extension peut être utilisée pour accéder à n’importe quelle version de MySQL, mais elle ne supporte pas les nouvelles fonctionnalités de la version 4.1 de MySQL.

Pour utiliser les nouvelles fonctionnalités de la version 4.1 de MySQL, il faut utiliser l’extension MySQLi, apparue en version 5 de PHP. Cette extension peut aussi être utilisée pour accéder à une version plus ancienne de MySQL, sous réserve de ne pas utiliser les fonctionnalités apparues dans la version 4.1 de MySQL.

Mis à part les nouvelles fonctionnalités de la version 4.1 de MySQL, les deux extensions sont très proches en termes de fonctionnalités. Très souvent, elles proposent les mêmes fonctions, avec des syntaxes identiques ou compatibles. Sauf exception, pour passer de l’utilisation de MySQL à MySQLi, il suffit de remplacer le préfixe `mysql_` dans le nom de la fonction par le préfixe `mysqli_`.

Nous commencerons par présenter de manière détaillée l’utilisation de l’extension MySQLi. Ensuite, nous établirons une correspondance entre l’extension MySQLi et l’extension MySQL, en se focalisant sur les principales différences.

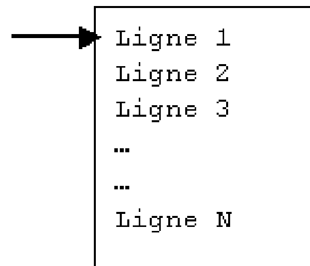
Nous présenterons aussi brièvement l’extension *PHP Data Objects* (PDO) qui définit une interface uniforme pour accéder

aux bases de données en PHP. Cette extension est apparue en version 5.1.

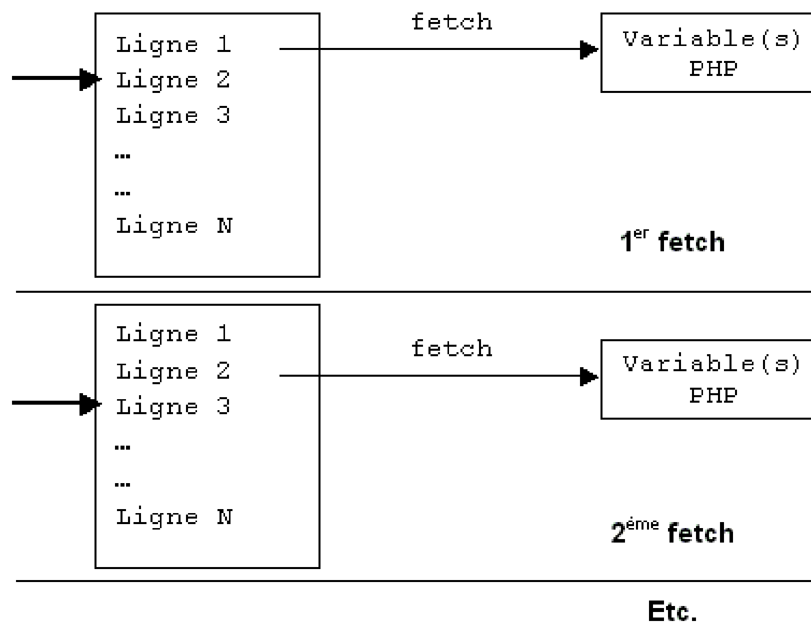
## 3. La notion de fetch

Dans la suite, nous verrons que l'instruction d'exécution d'une requête SELECT (en vue de lire des données) se contente d'exécuter la requête mais ne retourne aucune donnée. Après exécution de la requête, il faut extraire les lignes du résultat : c'est la notion de "fetch".

Pour résumer le fonctionnement, l'instruction d'exécution d'une requête SELECT identifie un résultat et positionne un pointeur interne sur la première ligne de ce résultat :




Une instruction supplémentaire permet de lire la ligne courante du résultat, de ramener les valeurs dans des variables PHP et de faire avancer le pointeur sur la ligne suivante :



# Utilisation de l'extension MySQLi

## 1. Introduction

L'extension MySQLi peut être utilisée soit sous une forme procédurale, soit sous une forme objet.

Dans sa forme orientée objet, l'extension MySQLi propose trois classes :

`mysqli`

Connexion entre PHP et MySQL.

`mysqli_stmt`

Requête préparée.

`mysqli_result`

Résultat de l'exécution d'une requête.

Ces différentes classes proposent des méthodes qui permettent d'effectuer les différentes actions (exécution d'une requête, récupération du résultat, etc.).

Dans sa forme procédurale, l'extension MySQLi propose des fonctions qui permettent d'effectuer les mêmes actions. De façon transparente, plusieurs de ces fonctions retournent ou acceptent en paramètres des objets de type `mysqli` ou `mysqli_result`.

Dans cet ouvrage, nous présenterons uniquement la forme procédurale de l'extension MySQLi.

La principale nouveauté de MySQL 4.1, utilisable avec l'extension MySQLi, est la notion de requête préparée.

Une requête préparée est une requête qui contient des paramètres matérialisés par un point d'interrogation (`?`).

*Exemples*

```
SELECT * FROM collection WHERE id = ?
INSERT INTO collection(nom,prix_ht) VALUES(?,?)
```

À l'inverse, une requête non préparée est une requête dans laquelle toutes les valeurs sont spécifiées.

*Exemples*

```
SELECT * FROM collection WHERE id = 1
INSERT INTO collection(nom,prix_ht) VALUES('TechNote',10.48)
```

Dans la suite de ce chapitre, nous présenterons comment exécuter des requêtes de lecture et de mise à jour, d'abord avec des requêtes non préparées (cf. dans cette section Utiliser des requêtes non préparées) puis avec des requêtes préparées (cf. dans cette section Utiliser des requêtes préparées).

## 2. Connexion et déconnexion

### a. Connexion

La fonction `mysqli_connect` permet d'établir une connexion avec une base MySQL.

*Syntaxe*

```
objet mysqli_connect([chaîne hôte [, chaîne utilisateur [,
chaîne mot_de_passe [, chaîne nom_base [, entier port ]]]]])
```

*Avec*

`hôte`

Nom (ou adresse IP) de l'hôte auquel il faut se connecter (machine locale par défaut).

`utilisateur`

Nom de l'utilisateur à utiliser pour établir la connexion. Valeur par défaut : le propriétaire du processus du serveur Web.

`mot_de_passe`

Mot de passe à utiliser pour établir la connexion. Valeur par défaut : chaîne vide (pas de mot de passe).

nom_base

Base de données MySQL sélectionnée par défaut (aucune par défaut).

port

Numéro du port pour la connexion au serveur MySQL (port standard 3306 par défaut).

La fonction mysqli_connect retourne un identifiant de connexion (objet mysqli) ou, en cas d'erreur, la valeur FALSE accompagnée d'un message d'alerte envoyé à l'affichage (cf. chapitre Gérer les erreurs dans un script PHP pour gérer cette situation correctement).

Plusieurs directives de configuration permettent de définir des valeurs par défaut pour les différents paramètres de la fonction mysqli_connect (voir la documentation).

### b. Déconnexion

Les connexions ouvertes dans un script sont automatiquement fermées à la fin du script, sauf déconnexion explicite avant avec la fonction mysqli_close.

La fonction mysqli_close permet de fermer une connexion en cours de script.

*Syntaxe*

```
booléen mysqli_close(objet connexion)
```

connexion

Identifiant de connexion retourné par la fonction mysqli_connect.

La fonction mysql_close retourne TRUE en cas de succès et FALSE en cas d'erreur (accompagné d'une alerte).

### c. Obtenir des informations sur le serveur MySQL

Les fonctions mysqli_get_host_info et mysqli_get_server_info permettent d'obtenir des informations sur le serveur MySQL.

*Syntaxe*

```
chaîne mysqli_get_host_info(objet connexion)
chaîne mysqli_get_server_info(objet connexion)
```

connexion

Identifiant de connexion retourné par la fonction mysqli_connect.

La fonction mysqli_get_host_info retourne des informations sur le type de connexion utilisée.

La fonction mysqli_get_server_info retourne la version du serveur MySQL.

### d. Obtenir des informations en cas d'erreur de connexion

Les fonctions mysqli_connect_errno et mysqli_connect_error permettent de récupérer des informations sur l'erreur éventuelle de la dernière connexion effectuée avec la fonction mysqli_connect.

*Syntaxe*

```
entier mysqli_connect_errno()
chaîne mysqli_connect_error()
```

La fonction mysqli_connect_errno retourne un numéro d'erreur (0 si aucune erreur) et la fonction mysqli_connect_error le message associé (chaîne vide si aucune erreur).

### e. Exemple

Dans cet exemple, nous utilisons l'opérateur @ pour ne pas afficher les alertes générées par les fonctions en cas d'erreur (cf. Gérer les erreurs dans un script PHP - Les fonctions de gestion des erreurs).

*Exemple*

```
<?php
// Définition d'une petite fonction qui ouvre une connexion.
function connecter($hôte,$utilisateur,$mot_de_passe) {
```

```
  $db = @mysqli_connect($hôte,$utilisateur,$mot_de_passe);
  if ($db) {
    echo 'Connexion réussie.<br />';
    echo 'Informations sur le serveur : ',
         mysqli_get_host_info($db),'<br />';
    echo 'Version du serveur : ',
         mysqli_get_server_info($db),'<br />';
  } else {
    printf(
      'Erreur %d : %s.<br />',
      mysqli_connect_errno(),mysqli_connect_error());
  }
  return $db;
}
// Définition d'une petite fonction qui ferme une connexion.
function déconnecter($connexion) {
  if ($connexion) {
    $ok = @mysqli_close($connexion);
    if ($ok) {
      echo 'Déconnexion réussie.<br />';
    } else {
      echo 'Echec de la déconnexion. <br />';
    }
  } else {
      echo 'Connexion non ouverte.<br />';
  }
}
// Premier test de connexion/déconnexion.
echo '<b>Premier test</b><br />';
$db = connecter('localhost','eniweb','web');
déconnecter($db);
// Deuxième test de connexion/déconnexion.
echo '<b>Deuxième test</b><br />';
$db = connecter('xampp','inconnu','inconnu');
déconnecter($db);
?>
```

*Résultat*

```
Premier test
Connexion réussie.
Informations sur le serveur : Localhost via UNIX socket
Version du serveur : 5.0.45
Déconnexion réussie.
Deuxième test
Erreur 1045 : Access denied for user 'inconnu'@'xampp' (using password: YES).
Connexion non ouverte.
```

## 3. Sélectionner une base de données

La fonction `mysqli_connect` présentée précédemment permet de sélectionner une base de données dès la connexion (quatrième paramètre de la fonction).

La fonction `mysqli_select_db` permet de sélectionner ou modifier la base de données à utiliser pour une connexion donnée.

*Syntaxe*

*booléen* `mysqli_select_db (`*objet* `connexion,` *chaîne* `nom_base)`

*Avec*

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

`nom_base`

Nom de la base de données.

La fonction `mysqli_select_db` retourne `TRUE` en cas de succès et `FALSE` en cas d'erreur. La fonction `mysqli_select_db` ne génère pas d'alerte en cas d'erreur.

*Exemple*

```
<?php
```

```
// Connexion.
$db = mysqli_connect('localhost','eniweb','web');
if (! $db) {
  exit('Echec de la connexion.');
}
echo 'Connexion réussie.<br />';
// Sélection de la base de données.
$ok = mysqli_select_db($db,'eni');
if ($ok) {
  echo 'Base de données sélectionnée.<br />';
} else {
  echo 'Echec de la sélection de la base de données.';
}
// Déconnexion.
$ok = mysqli_close($db);
if ($ok) {
  echo 'Déconnexion réussie.';
} else {
  echo 'Echec de la déconnexion.';
}
?>
```

*Résultat*

```
Connexion réussie.
Base de données sélectionnée.
Déconnexion réussie.
```

## 4. Utiliser des requêtes non préparées

### a. Vue d'ensemble

Les étapes d'utilisation d'une requête non préparée sont les suivantes :

| **Requête de lecture (`SELECT`)** | **Requête de mise à jour (`INSERT`, `UPDATE`, `DELETE`)** |
|---|---|
| Exécuter la requête = `mysqli_query` | |
| Connaître le nombre de lignes dans le résultat = `mysqli_num_rows` | Connaître le nombre de lignes traitées = `mysqli_affected_rows` |
| Extraire les lignes du résultat = `mysqli_fetch_array` ou `mysqli_fetch_assoc` ou `mysqli_fetch_object` ou `mysqli_fetch_row`. | Connaître la valeur du dernier identifiant généré pour une colonne ayant le type `AUTO_INCREMENT` = `mysqli_insert_id` |

En complément, les fonctions `mysqli_errno` et `mysqli_error` permettent de récupérer des informations sur l'erreur éventuelle de la dernière opération effectuée dans une session.

### b. Exécuter une requête

La fonction `mysqli_query` permet d'exécuter une requête sur une base de données.

*Syntaxe*

```
objet mysqli_query(objet connexion, chaîne requête [, entier mode])
```

*Avec*

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

`requête`

Texte de la requête à exécuter.

`mode`

Indique si le résultat doit être mis en buffer (constante `MYSQLI_STORE_RESULT`, valeur par défaut) ou non (constante `MYSQLI_USE_RESULT`).

Dans le cas d'une requête SQL qui retourne un résultat (comme SELECT, SHOW ou DESCRIBE), la fonction mysqli_query retourne un identifiant de résultat de requête en cas de succès (objet mysqli_result) et FALSE en cas d'échec. Pour les autres ordres SQL, cette fonction retourne TRUE en cas de succès et FALSE en cas d'échec. La fonction mysqli_query ne génère pas d'alerte en cas d'erreur.

Dans le cas d'une requête SQL qui retourne un résultat, la fonction mysqli_query exécute la requête, indique si la requête s'est exécutée correctement mais ne renvoie aucune donnée. Il va falloir extraire les lignes du résultat. Le troisième paramètre de la fonction mysqli_query indique si le résultat de la requête est mis en buffer (constante MYSQLI_STORE_RESULT, valeur par défaut) ou non (MYSQLI_USE_RESULT). Pour les requêtes volumineuses, l'utilisation de MYSQLI_USE_RESULT consomme beaucoup moins de mémoire et permet d'extraire la première ligne beaucoup plus rapidement : il n'y a pas besoin d'attendre que la totalité du résultat soit mis en buffer. Par contre, après exécution de la requête avec MYSQLI_USE_RESULT, il n'est pas possible de connaître le nombre de lignes du résultat, ni d'exécuter une autre requête avant d'avoir extrait la totalité des lignes ou libéré le résultat (fonction mysqli_free_result).

*Exemple (avec une requête SELECT)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Exécution d'une requête SELECT.
$requête = mysqli_query($db,'SELECT * FROM collection');
if ($requête === FALSE) {
  echo 'Echec de l\'exécution de la requête';
} else {
  echo 'Exécution réussie.';
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Exécution réussie.
```

### c. Connaître le nombre de lignes dans le résultat d'une requête de lecture

La fonction mysqli_num_rows permet de connaître le nombre de lignes dans le résultat.

*Syntaxe*

```
entier mysqli_num_rows(objet résultat)
```

*Avec*

résultat

Identifiant de résultat de requête retourné par la fonction mysqli_query.

Cette fonction est utilisable uniquement pour les requêtes qui retournent un résultat (comme SELECT, SHOW ou DESCRIBE) et pour lesquelles le résultat a été mis en buffer (voir la fonction mysqli_query).

*Exemple*

```
<?php
// Connexion (avec sélection de la base de données).
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Exécution d'une requête
$requête = mysqli_query($db,'SELECT * FROM collection');
if ($requête === FALSE) {
  echo 'Echec de l\'exécution de la requête';
} else {
  // Affichage du nombre de lignes dans le résultat
  echo 'Nombre de collections : ',mysqli_num_rows($requête);
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Nombre de collections : 4
```

### d. Extraire le résultat d'une requête de lecture

Le résultat de l'exécution d'une requête qui retourne un résultat (comme `SELECT`, `SHOW` ou `DESCRIBE`) peut être lu par les fonctions `mysqli_fetch_array`, `mysqli_fetch_assoc`, `mysqli_fetch_object` ou `mysqli_fetch_row`.

Ces fonctions font la même chose : elles lisent la ligne courante du résultat et font avancer le pointeur sur la ligne suivante (voir l'introduction).

Ces fonctions diffèrent sur le type de donnée utilisé pour retourner le résultat.

*Syntaxe*

```
tableau mysqli_fetch_array(objet résultat[, entier type])
tableau mysqli_fetch_assoc(objet résultat)
objet mysqli_fetch_object(objet résultat)
tableau mysqli_fetch_row(objet résultat)
```

*Avec*

`résultat`

Identifiant de résultat de requête retourné par la fonction `mysqli_query`.

`type`

Type de résultat égal à une des constantes suivantes : `MYSQLI_ASSOC`, `MYSQLI_NUM`, `MYSQLI_BOTH` (valeur par défaut).

Les fonctions `mysqli_fetch_array`, `mysqli_fetch_assoc` et `mysqli_fetch_row` retournent la ligne courante du résultat sous la forme d'un tableau, chaque ligne du tableau correspondant à une colonne du résultat. La fonction `mysqli_fetch_object` retourne la ligne courante sous la forme d'un objet.

S'il n'y a plus de ligne à lire dans le résultat, ces fonctions retournent `NULL`.

Pour la fonction `mysqli_fetch_assoc`, le tableau est un tableau associatif dont la clé est le nom de la colonne. Pour la fonction `mysqli_fetch_row`, il s'agit d'un tableau à indices entiers, l'indice 0 correspondant à la première colonne, l'indice 1 à la deuxième, etc. Enfin pour la fonction `mysqli_fetch_array`, le type du tableau dépend du deuxième paramètre :

`MYSQLI_NUM`

Tableau à indices entiers (comme la fonction `mysqli_fetch_row`).

`MYSQLI_ASSOC`

Tableau associatif (comme la fonction `mysqli_fetch_assoc`).

`MYSQLI_BOTH` (valeur par défaut)

Les deux à la fois. Chaque colonne est présente deux fois, une fois avec un indice entier correspondant à sa position et une fois avec une clé correspondant à son nom.

*Exemple*

| | | | |
|---|---|---|---|
| **Requête** | SELECT id,nom,prix_ht FROM collection | | |
| **Colonnes** | id | nom | prix_ht |
| **1ère ligne du résultat** | 1 | Ressources Informatiques | 24.44 |

*Résultat d'un fetch avec :*

| mysqli_fetch_row ou mysqli_fetch_array (…,MYSQL_NUM) | | mysqli_fetch_assoc ou mysqli_fetch_array (…,MYSQL_ASSOC) | | mysqli_fetch_array (…,MYSQL_BOTH) | |
|---|---|---|---|---|---|
| **Clé** | **Valeur** | **Clé** | **Valeur** | **Clé** | **Valeur** |
| 0 | 1 | id | 1 | id | 1 |
| 1 | Ressources Informatiques | nom | Ressources Informatiques | 0 | 1 |
| 2 | 24.44 | prix_ht | 24.44 | nom | Ressources Informatiques |
| | | | | 1 | Ressources Informatiques |
| | | | | prix_ht | 24.44 |
| | | | | 2 | 24.44 |

La fonction mysqli_fetch_object retourne un objet, avec un attribut par colonne, le nom de l'attribut correspondant au nom de la colonne.

*Exemple*

| Attribut | Valeur |
|---|---|
| id | 1 |
| nom | Ressources Informatiques |
| prix_ht | 24.44 |

En cas d'utilisation, d'un alias de colonne dans la requête SELECT (exemple SELECT AVG(prix_ht) **prix_moyen**FROM collection), c'est l'alias de colonne qui est utilisé comme clé ou nom d'attribut.

*Exemple*

```
<?php
// Inclusion du fichier qui contient la définition de
// la fonction 'afficher_tableau'.
require('fonctions.inc');
// Connexion (avec sélection de la base de données).
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Exécution d'une requête
$sql = 'SELECT id,nom,prix_ht FROM collection LIMIT 4';
$requête = mysqli_query($db,$sql);
// Premier fetch avec mysqli_fetch_row.
$ligne = mysqli_fetch_row($requête);
afficher_tableau($ligne,'mysql_fetch_row');
// Deuxième fetch avec mysql_fetch_assoc.
$ligne = mysqli_fetch_assoc($requête);
afficher_tableau($ligne,'mysql_fetch_assoc');
// Troisième fetch avec mysql_fetch_array :
// -> sans deuxième paramètre = MYSQLI_BOTH
$ligne = mysqli_fetch_array($requête);
afficher_tableau($ligne,'mysql_fetch_array');
// Quatrième fetch avec mysql_fetch_object.
$ligne = mysqli_fetch_object($requête);
echo "<p /><b>mysql_fetch_object</b><br />";
echo "\$ligne->id = $ligne->id<br />";
echo "\$ligne->nom = $ligne->nom<br />";
echo "\$ligne->prix_ht = $ligne->prix_ht<br />";
// Cinquième fetch de nouveau avec mysql_fetch_row :
// -> normalement, plus de ligne.
$ligne = mysqli_fetch_row($requête);
if ($ligne === NULL) {
      echo '<p /><b>Cinquième fetch : plus rien</b>';
}
```

```
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
mysql_fetch_row
0 = 1
1 = Ressources Informatiques
2 = 24.44
mysql_fetch_assoc
id = 2
nom = TechNote
prix_ht = 9.48
mysql_fetch_array
0 = 3
id = 3
1 = Les TP Informatiques
nom = Les TP Informatiques
2 = 25.59
prix_ht = 25.59
mysql_fetch_object
$ligne->id = 4
$ligne->nom = Coffret Technique
$ligne->prix_ht = 46.45
Cinquième fetch : plus rien
```

En cas d'utilisation d'un identifiant de résultat non valide, les fonctions `mysqli_fech_*` retournent `NULL` et affichent une alerte (cf. Gérer les erreurs).

Toutes les méthodes se valent, notamment du point de vue des performances. Les fonctions `mysqli_fetch_assoc` et `mysqli_fetch_object` permettent d'employer le nom des colonnes de la requête et de rendre le code plus lisible.

*Exemple de lecture de la totalité du résultat*

```
<?php
// Connexion (avec sélection de la base de données).
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Exécution d'une requête
$sql = 'SELECT id,titre FROM livre WHERE id_collection = 1';
$requête = mysqli_query($db,$sql);
// Lecture du résultat.
while ($ligne = mysqli_fetch_assoc($requête)) {
  echo $ligne['id'],' - ',$ligne['titre'],'<br />';
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
1 - PHP 4
2 - PHP 5.2
4 - Oracle 10g
7 - BusinessObjects 6
8 - MySQL 5
```

Pour obtenir des informations sur les colonnes du résultat (nom de table, nom de colonne, type de données, etc.), vous pouvez utiliser les fonctions `mysqli_fetch_field` ou `mysqli_fetch_fields`. Pour en savoir plus, consultez la documentation.

### e. Obtenir des informations sur le résultat d'une requête de mise à jour

Comme nous l'avons indiqué en préambule, mettre à jour des données consiste à exécuter des requêtes `INSERT` (création), `UPDATE` (modification) ou `DELETE` (suppression) à l'aide de la fonction `mysqli_query`, comme pour une requête `SELECT`.

En complément, deux fonctions sont intéressantes : `mysqli_affected_rows` et `mysqli_insert_id`.

La fonction `mysqli_affected_rows` permet de connaître le nombre de lignes concernées (insérées, modifiées ou supprimées) par la dernière requête `INSERT`, `UPDATE` ou `DELETE` exécutée dans une session.

*Syntaxe*

```
entier mysqli_affected_rows(objet connexion)
```

*Avec*

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

Si la dernière requête a échoué, la fonction `mysql_affected_rows` retourne -1.

---

Pour une requête de sélection, la fonction `mysqli_affected_rows` donne le même résultat que la fonction `mysqli_num_rows`.

---

Dans le cas d'un ordre `UPDATE`, `mysqli_affected_rows` ne compte pas les lignes non modifiées lorsque les valeurs avant et après sont les mêmes.

---

La fonction `mysqli_insert_id` retourne la valeur du dernier identifiant généré pour une colonne ayant le type `AUTO_INCREMENT` par une requête `INSERT` dans une session.

*Syntaxe*

```
entier mysqli_insert_id(objet connexion)
```

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

Si aucun identifiant n'a été généré automatiquement par la dernière requête, la fonction `mysqli_insert_id` retourne 0.

*Exemples*

```
<?php
// Définition d'une petite fonction d'affichage de la liste
// des collections.
function afficher_collections($db) {
  $sql = 'SELECT * FROM collection';
  $requête = mysqli_query($db,$sql);
  echo "<b>Liste des collections :</b><br />";
  while ($ligne = mysqli_fetch_assoc($requête)) {
    echo $ligne['id'],' - ',$ligne['nom'],
         ' - ',$ligne['prix_ht'],'<br />';
  }
}
// Connexion (avec sélection de la base de données).
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Affichage de contrôle.
afficher_collections ($db);
 // Requête INSERT.
$sql = "INSERT INTO collection(nom,prix_ht) " .
       "VALUES('Coffret Solutions',55.92)";
$requête = mysqli_query($db,$sql);
$identifiant = mysqli_insert_id($db);
echo 'Identifiant de la nouvelle collection = ',
     $identifiant,'<br />';
// Requête UPDATE.
$sql = "UPDATE collection SET prix_ht = prix_ht * 1.05 " .
       "WHERE prix_ht < 25";
$requête = mysqli_query($db,$sql);
$nombre = mysqli_affected_rows($db);
echo "$nombre collections(s) augmentée(s).<br />";
// Affichage de contrôle.
afficher_collections($db);
?>
```

*Résultat*

```
Liste des collections :
1 - Ressources Informatiques - 24.44
2 - TechNote - 9.48
3 - Les TP Informatiques - 25.59
4 - Coffret Technique - 46.45
```

```
Identifiant de la nouvelle collection = 5
2 collections(s) augmentée(s).
Liste des collections :
1 - Ressources Informatiques - 25.66
2 - TechNote - 9.95
3 - Les TP Informatiques - 25.59
4 - Coffret Technique - 46.45
5 - Coffret Solutions - 55.92
```

### f. Gérer les erreurs

Les fonctions `mysqli_errno` et `mysqli_error` permettent de récupérer des informations sur l'erreur éventuelle de la dernière opération effectuée dans une session.

*Syntaxe*

```
entier mysqli_errno(objet connexion)
chaîne mysqli_error(objet connexion)
```

*Avec*

connexion

Identifiant de connexion retourné par la fonction `mysqli_connect`.

La fonction `mysqli_errno` retourne un numéro d'erreur (0 si aucune erreur) et la fonction `mysqli_error` le message associé (chaîne vide si aucune erreur).

*Exemple*

```
<?php
// Connexion.
$db = mysqli_connect('localhost','eniweb','web');
if (! $db) {
  exit('Echec de la connexion.');
}
// Sélection d'une mauvaise base de données.
$ok = mysqli_select_db($db,'hermes');
echo '1 : ',mysqli_errno($db),' - ',
            mysqli_error($db),'<br />';
// Sélection de la bonne base de données.
$ok = mysqli_select_db($db,'eni');
// Requête sur une table qui n'existe pas.
$sql = 'SELECT * FROM article';
$requête = mysqli_query($db,$sql);
echo '2 : ',mysqli_errno($db),' - ',
            mysqli_error($db),'<br />';
// Fetch sur un mauvais résultat.
$ligne = mysqli_fetch_assoc($requête);
echo '3 : ',mysqli_errno($db),' - ',
            mysqli_error($db),'<br />';
// Requête INSERT qui viole une clé unique.
$sql = "UPDATE collection SET nom = 'TechNote' WHERE id = 1";
$requête = mysqli_query($db,$sql);
echo '4 : ',mysqli_errno($db),' - ',
            mysqli_error($db),'<br />';
?>
```

*Résultat*

```
1 : 1044 - Access denied for user 'eniweb'@'localhost' to database 'hermes'
2 : 1146 - Table 'eni.article' doesn't exist
Warning: mysqli_fetch_assoc() expects parameter 1 to be
mysqli_result, boolean given in /app/scripts/index.php on line 19
3 : 1146 - Table 'eni.article' doesn't exist
4 : 1062 - Duplicate entry 'TechNote' for key 2
```

Les points 2 et 3 illustrent le fait que les erreurs liées à l'utilisation d'une ressource (de connexion ou de résultat) non valide génèrent une alerte qui est directement affichée. Dans ce cas, l'erreur n'est pas une erreur MySQL et les fonctions `mysqli_errno` ou `mysqli_error` ne sont pas réinitialisées et ne retournent donc pas d'erreur spécifique : le message du point 3 est en fait celui du point 2.

En pratique, les fonctions `mysqli_errno` et `mysqli_error` sont utilisées après l'exécution des requêtes.

## 5. Utiliser des requêtes préparées

### a. Vue d'ensemble

La principale nouveauté de MySQL 4.1, utilisable avec l'extension MySQLi, est la notion de requête préparée.

Une requête préparée est une requête qui contient des paramètres matérialisés par un point d'interrogation (?).

*Exemples*

```
SELECT * FROM articles WHERE identifiant = ?
INSERT INTO articles(libelle,prix) VALUES(?,?)
```

Un paramètre ne peut pas remplacer un nom de table, un nombre de colonne ou toute une partie de la requête.

*Exemples interdits*

```
SELECT * FROM ?
SELECT * FROM articles WHERE ?
```

Les étapes d'utilisation d'une requête préparée sont les suivantes :

| **Requête de lecture (SELECT)** | **Requête de mise à jour (INSERT, UPDATE, DELETE)** |
|---|---|
| Préparer la requête = mysqli_prepare | |
| Lier des variables PHP aux paramètres de la requête = mysqli_stmt_bind_param | |
| Exécuter la requête = mysqli_stmt_execute | |
| Connaître le nombre de lignes dans le résultat = mysqli_num_rows | Connaître le nombre de lignes traitées = mysqli_stmt_affected_rows |
| Lier des variables PHP aux colonnes du résultat = mysqli_stmt_bind_result | Connaître la valeur du dernier identifiant généré pour une colonne ayant le type AUTO_INCREMENT = mysqli_stmt_insert_id |
| Extraire les lignes du résultat = mysqli_stmt_fetch | |
| Fermer la requête préparée = mysqli_stmt_close | |

Dans le cas d'une requête de type SELECT, il est possible de stocker dans un buffer le résultat complet de l'exécution d'une requête préparée. Dans ce cas, les fonctions suivantes sont utiles :

mysqli_stmt_store_result

Stocke dans un buffer le résultat complet d'une requête préparée de type SELECT.

mysqli_stmt_num_rows

Retourne le nombre de lignes sélectionnées par une requête préparée de type SELECT, dont le résultat a été stocké au préalable avec mysqli_stmt_store_result.

mysqli_stmt_free_result

Libère le résultat d'une requête préparée de type SELECT, dont le résultat a été stocké au préalable avec mysqli_stmt_store_result.

En complément, les fonctions mysqli_stmt_errno et mysqli_stmt_error permettent de récupérer des informations sur l'erreur éventuelle de la dernière exécution d'une requête préparée.

À chaque exécution de la requête préparée, la valeur courante des variables PHP associées aux paramètres est utilisée. L'intérêt est de pouvoir employer plusieurs fois la même requête avec des valeurs différentes des paramètres sans analyser de nouveau la requête, ce qui permet d'améliorer les performances.

### b. Préparer une requête

La fonction mysqli_prepare prépare une requête pour l'exécution.

*Syntaxe*

```
objet mysqli_prepare(objet connexion, chaîne sql)
```

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

`sql`

Texte de la requête SQL.

La fonction `mysqli_prepare` retourne une ressource de requête préparée (objet `mysqli_stmt`) ou, en cas d'erreur, la valeur `NULL` accompagnée d'un message d'alerte envoyé à l'affichage (cf. chapitre Gérer les erreurs dans un script PHP pour gérer cette situation correctement).

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'INSERT INTO collection(nom,prix_ht) VALUES(?,?)';
$requête = mysqli_prepare($db, $sql);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

Les fonctions `mysqli_stmt_init` et `mysqli_stmt_prepare` permettent de faire la même chose, mais en deux étapes.

### c. Lier des variables PHP aux paramètres de la requête

La fonction `mysqli_stmt_bind_param` lie des variables aux paramètres d'une requête préparée.

*Syntaxe*

```
booléen mysqli_stmt_bind_param(objet requête, chaîne types,
mixte variable[, ...])
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

`types`

Chaîne de caractères qui contient un ou plusieurs caractères qui spécifient le type de données de la variable à lier : `i` = variable de type entier ; `d` = variable de type nombre décimal ; `s` = variable de type chaîne de caractères ; `b` = variable de type blob.

`variable`

Variable à lier à un paramètre.

La fonction `mysqli_stmt_bind_param` retourne `TRUE` en cas de succès ou, en cas d'erreur, la valeur `FALSE` accompagnée d'un message d'alerte envoyé à l'affichage (cf. chapitre Gérer les erreurs dans un script PHP pour gérer cette situation correctement).

Il doit y avoir exactement le même nombre de variables, et donc de caractères dans le paramètre `types`, que de paramètres dans la requête. L'association est positionnelle (première variable pour le premier `?`, etc.).

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'INSERT INTO collection(nom,prix_ht) VALUES(?,?)';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'sd',$nom,$prix_ht);
```

```
// Déconnexion.
$ok = mysqli_close($db);
?>
```

Comme le montre cet exemple, les variables liées n'ont pas besoin d'être définies au moment de l'appel à la fonction mysqli_stmt_bind_param.

Les paramètres peuvent être liés avec des lignes d'un tableau ou les attributs d'un objet. Dans les deux cas, mysqli_stmt_bind_param crée le tableau ou instancie l'objet, s'ils n'existent pas déjà. Dans ce cas, il ne faut pas de nouveau créer le tableau, ou instancier l'objet, après l'appel à mysqli_stmt_bind_param sous peine de "casser" la liaison.

*Exemple*

```
<?php
...
// Liaison des paramètres.
// Avec les lignes d'un tableau.
$ok = mysqli_stmt_bind_param
        ($requête,'sd',
         $collection['nom'],$collection['prix_ht']);
...
?>
```

```
<?php
...
// Liaison des paramètres.
// Avec les attributs d'un objet.
$ok = mysqli_stmt_bind_param
        ($requête,'sd',
         $collection->nom,$collection->prix_ht);
...
?>
```

### d. Exécuter la requête préparée

La fonction mysqli_stmt_execute exécute une requête préparée.

*Syntaxe*

```
booléen mysqli_stmt_execute(objet requête)
```

requête

Ressource de requête préparée retournée par la fonction mysqli_prepare (ou mysqli_stmt_init).

La fonction mysqli_execute retourne TRUE en cas de succès ou FALSE en cas d'erreur.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'INSERT INTO collection(nom,prix_ht) VALUES(?,?)';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'sd',$nom,$prix_ht);
// Exécution de la requête.
$nom = 'Epsilon';
$prix_ht = 51.18;
$ok = mysqli_stmt_execute($requête);
// Nouvelle exécution de la requête.
$nom = 'Expert IT';
$prix_ht = 36.97;
$ok = mysqli_stmt_execute($requête);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

### e. Obtenir des informations sur le résultat d'une requête de mise à jour

La fonction mysqli_stmt_affected_rows retourne le nombre de lignes mises à jour par une requête préparée.

*Syntaxe*

```
entier mysqli_stmt_affected_rows(objet requête)
```

requête

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

Si la requête a échoué, ou si elle n'est pas une requête de mise à jour, la fonction `mysqli_stmt_affected_rows` retourne -1.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'UPDATE collection SET frais_ht = ? ' .
       'WHERE frais_ht IS NULL';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'d',$frais_ht);
// Exécution de la requête.
$frais_ht = 1;
$ok = mysqli_stmt_execute($requête);
echo 'Nombre de collection(s) modifiée(s) = ',
     mysqli_stmt_affected_rows ($requête),'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Nombre de collection(s) modifiée(s) = 5
```

La fonction`mysqli_stmt_insert_id` retourne la valeur du dernier identifiant généré, pour une colonne ayant le type `AUTO_INCREMENT`, par une requête préparée `INSERT`.

*Syntaxe*

```
entier mysqli_stmt_insert_id(objet requête)
```

requête

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

Si aucun identifiant n'a été généré automatiquement par la dernière requête, la fonction `mysqli_stmt_insert_id` retourne 0.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'INSERT INTO collection(nom) VALUES(?)';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'s',$nom);
// Exécution de la requête.
$nom = 'Solution Informatiques';
$ok = mysqli_stmt_execute($requête);
echo 'Identifiant de la nouvelle collection = ',
     mysqli_stmt_insert_id($requête),'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Identifiant de la nouvelle collection = 8
```

### f. Lier des variables PHP aux colonnes du résultat d’une requête de lecture

La fonction `mysqli_stmt_bind_result` associe des variables PHP aux colonnes du résultat d’une requête préparée de type `SELECT`.

*Syntaxe*

```
booléen mysqli_stmt_bind_result(objet requête, mixte variable [, ...])
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

`variable`

Variable à lier à une colonne du résultat.

La fonction `mysqli_stmt_bind_result` retourne `TRUE` en cas de succès ou, en cas d’erreur, la valeur `FALSE` accompagnée d’un message d’alerte envoyé à l’affichage (cf. chapitre Gérer les erreurs dans un script PHP pour gérer cette situation correctement).

Il doit y avoir exactement le même nombre de variables que de colonnes dans le résultat. L’association est positionnelle (première variable pour la première colonne, etc.).

L’appel à la fonction `mysqli_stmt_bind_result` est nécessaire pour toutes les requêtes qui retourne un résultat (`SELECT`, `SHOW`, `DESCRIBE`).

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'SELECT id,titre FROM livre WHERE id_collection = ?';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'i',$id_collection);
// Exécution de la requête.
$id_collection = 1;
$ok = mysqli_stmt_execute($requête);
// Liaison des colonnes du résultat.
$ok = mysqli_stmt_bind_result($requête,$id,$titre);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

Les colonnes du résultat peuvent être liées avec des lignes d’un tableau ou les attributs d’un objet. Dans les deux cas, `mysqli_stmt_bind_result` crée le tableau ou instancie l’objet, s’ils n’existent pas déjà. Dans ce cas, il ne faut pas de nouveau créer le tableau, ou instancier l’objet, après l’appel à `mysqli_stmt_bind_result` sous peine de "casser" la liaison.

*Exemple*

```
<?php
...
// Liaison des colonnes du résultat.
// Avec un tableau.
$ok = mysqli_stmt_bind_result
        ($requête,$livre['id'],$livre['titre']);
...
?>
```

```
<?php
...
// Liaison des colonnes du résultat.
// Avec un tableau, en laissant PHP affecter les indices.
$ok = mysqli_stmt_bind_result
        ($requête,$livre[],$livre[]);
...
?>
```

```
<?php
...
// Liaison des colonnes du résultat.
// Avec les attributs d'un objet.
$ok = mysqli_stmt_bind_result
        ($requête,$livre->id,$livre->titre);
```

```
...
?>
```

### g. Extraire le résultat d'une requête de lecture

La fonction mysqli_stmt_fetch lit une ligne de résultat d'une requête préparée de type SELECT (SELECT, SHOW, DESCRIBE).

*Syntaxe*

```
booléen mysqli_stmt_fetch(objet requête)
```

requête

Ressource de requête préparée retournée par la fonction mysqli_prepare (ou mysqli_stmt_init).

La fonction mysqli_stmt_fetch retourne TRUE en cas de succès, FALSE en cas d'erreur et NULL s'il n'y a plus de ligne à lire.

La ligne est extraite dans les variables PHP liées au préalable au résultat.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'SELECT id,titre FROM livre WHERE id_collection = ?';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'i',$id_collection);
// Exécution de la requête.
$id_collection = 1;
$ok = mysqli_stmt_execute($requête);
// Liaison des colonnes du résultat.
$ok = mysqli_stmt_bind_result($requête,$id,$titre);
// Lecture du résultat.
echo "<b>Collection numéro $id_collection</b><br />";
while (mysqli_stmt_fetch($requête)) {
  echo "$id - $titre<br />";
}
// Nouvelle exécution et lecture du résultat
// (inutile de refaire les liaisons).
$id_collection = 3;
$ok = mysqli_stmt_execute($requête);
echo "<b>Collection numéro $id_collection</b><br />";
while (mysqli_stmt_fetch($requête)) {
  echo "$id - $titre<br />";
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Collection numéro 1
1 - PHP 4
2 - PHP 5.2
4 - Oracle 10g
7 - BusinessObjects 6
8 - MySQL 5
Collection numéro 3
9 - PHP et MySQL (versions 4 et 5)
```

### h. Utiliser un résultat stocké

Dans le cas d'une requête de type SELECT, il est possible de stocker dans un buffer le résultat complet de l'exécution d'une requête préparée. Dans ce cas, les fonctions suivantes sont utiles :

mysqli_stmt_store_result

Stocke dans un buffer le résultat complet d'une requête préparée de type SELECT.

`mysqli_stmt_num_rows`

Retourne le nombre de lignes sélectionnées par une requête préparée de type `SELECT`, dont le résultat a été stocké au préalable avec `mysqli_stmt_store_result`.

`mysqli_stmt_free_result`

Libère le résultat d'une requête préparée de type `SELECT`, dont le résultat a été stocké au préalable avec `mysqli_stmt_store_result`.

Le seul intérêt de stocker le résultat est de pouvoir connaitre immédiatement le nombre de lignes dans le résultat (avec la fonction `mysqli_stmt_num_rows`). Par contre, stocker le résultat d'une requête qui retourne un grand nombre de lignes consomme de la mémoire (côté client).

La fonction `mysqli_stmt_fetch` présentée précédemment permet de lire le résultat stocké (sans différence de syntaxe).

## mysqli_stmt_store_result

La fonction `mysqli_stmt_store_result` stocke dans un buffer le résultat complet d'une requête préparée de type `SELECT` (`SELECT`, `SHOW`, `DESCRIBE`).

*Syntaxe*

```
booléen mysqli_stmt_store_result(objet requête)
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

La fonction `mysqli_stmt_store_result` retourne `TRUE` en cas de succès et `FALSE` en cas d'erreur.

## mysqli_stmt_num_rows

La fonction `mysqli_stmt_num_rows` retourne le nombre de lignes sélectionnées par une requête préparée de type `SELECT`, dont le résultat a été stocké au préalable avec la fonction `mysqli_stmt_store_result`.

*Syntaxe*

```
entier mysqli_stmt_num_rows(objet requête)
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

La fonction `mysqli_stmt_bind_result` retourne toujours 0 si le résultat n'a pas été stocké au préalable à l'aide de la fonction `mysqli_stmt_store_result`.

## mysqli_stmt_free_result

La fonction `mysqli_stmt_free_result` libère le résultat d'une requête préparée de type `SELECT`, dont le résultat a été stocké au préalable avec result.

*Syntaxe*

```
mysqli_stmt_free_result(objet requête)
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

La fonction `mysqli_stmt_free_result` ne fait rien si le résultat n'a pas été stocké au préalable à l'aide de la fonction `mysqli_stmt_store_result`.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation de la requête.
$sql = 'SELECT id,titre FROM livre WHERE id_collection = ?';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'i',$id_collection);
// Exécution de la requête.
$id_collection = 1;
```

```
$ok = mysqli_stmt_execute($requête);
// Liaison des colonnes du résultat.
$ok = mysqli_stmt_bind_result($requête,$id,$titre);
echo '<b>Avant appel à mysqli_stmt_store_result</b><br />',
     'Nombre de lignes sélectionnées = ',
     mysqli_stmt_num_rows($requête),'<br />';
$ok = mysqli_stmt_store_result($requête);
echo '<b>Après appel à mysqli_stmt_store_result</b><br />',
     'Nombre de lignes sélectionnées = ',
     mysqli_stmt_num_rows($requête),'<br />';
// Lecture du résultat.
echo "<b>Collection numéro $id_collection</b><br />";
while (mysqli_stmt_fetch($requête)) {
  echo "$id - $titre<br />";
}
// Libération du résultat.
mysqli_stmt_free_result($requête);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Avant appel à mysqli_stmt_store_result
Nombre de lignes sélectionnées = 0
Après appel à mysqli_stmt_store_result
Nombre de lignes sélectionnées = 5
Collection numéro 1
1 - PHP 4
2 - PHP 5.2
4 - Oracle 10g
7 - BusinessObjects 6
8 - MySQL 5
```

### i. Gérer les erreurs

Les fonctionsmysqli_stmt_errno et mysqli_stmt_error permettent de récupérer des informations sur l'erreur éventuelle de la dernière exécution d'une requête préparée.

*Syntaxe*

```
entier mysqli_stmt_errno(objet requête)
chaîne mysqli_stmt_error(objet requête)
```

requête

Ressource de requête préparée retournée par la fonction mysqli_prepare (ou mysqli_stmt_init).

La fonction mysqli_stmt_errno retourne un numéro d'erreur (0 si aucune erreur) et la fonction mysqli_stmt_error le message associé (chaîne vide si aucune erreur).

Pour pouvoir utiliser ces deux fonctions, il faut passer en paramètre une ressource de requête préparée valide. Cela signifie que la requête doit avoir été préparée avec succès avec la fonction mysqli_prepare. Pour obtenir des informations sur une erreur de préparation, il faut utiliser les fonctions mysqli_errno et/ou mysqli_error (cf. dans cette section Utiliser des requêtes non préparées - Gérer les erreurs).

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Préparation d'une requête sur une table qui n'existe pas.
$sql = 'SELECT * FROM article';
$requête = mysqli_prepare($db, $sql);
// Utilisation de mysqli_errno et mysqli_error à ce stade.
echo '1 : ',mysqli_errno($db),' - ',
            mysqli_error($db),'<br />';
// Préparation d'une requête (sur une table qui existe).
$sql = 'UPDATE collection SET nom = ? WHERE id = ?';
$requête = mysqli_prepare($db, $sql);
// Liaison des paramètres.
$ok = mysqli_stmt_bind_param($requête,'si',$nom,$id);
// Exécution de la requête (viole une clé unique).
```

```
$id = 1;
$nom = 'TechNote';
$ok = mysqli_stmt_execute($requête);
echo '2 : ',mysqli_stmt_errno($requête),' - ',
            mysqli_stmt_error($requête),'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
1 : 1146 - Table 'eni.article' doesn't exist
2 : 1062 - Duplicate entry 'TechNote' for key 2
```

### j. Fermer une requête préparée

La fonction `mysqli_stmt_close` ferme une requête préparée.

*Syntaxe*

```
booléen mysqli_stmt_close(objet requête)
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

La fonction `mysqli_stmt_close` retourne `TRUE` en cas de succès, ou, en cas d'erreur, la valeur `NULL` accompagnée d'un message d'alerte envoyé à l'affichage (cf. chapitre Gérer les erreurs dans un script PHP pour gérer cette situation correctement). La fonction retourne une erreur si la requête n'a pas été préparée (avec succès).

 Il est conseillé de fermer une requête préparée avant de réutiliser la variable pour une autre requête.

## 6. Appeler un programme stocké

### a. Procédure stockée

Pour appeler une procédure stockée, il faut exécuter l'ordre SQL `CALL nom_procédure(...)` à l'aide des fonctions `mysqli_query` ou `mysqli_stmt_execute`.

Si la procédure stockée retourne un résultat directement (utilisation d'un ordre `SELECT` par exemple), ce dernier doit être lu à l'aide des fonctions de fetch adaptées.

Si la procédure stockée possède un paramètre `OUT`, il faut utiliser une variable MySQL (`@variable`) dans l'ordre SQL `CALL`, puis exécuter une requête `SELECT @variable` pour récupérer le résultat dans le script PHP.

**Premier exemple : procédure avec paramètre OUT**

Dans ce premier exemple, nous appellerons une procédure stockée qui permet de créer une nouvelle collection et qui retourne l'identifiant de la nouvelle collection dans un paramètre `OUT`.

*Code source de la procédure stockée*

```
CREATE PROCEDURE ps_creer_collection
  (
  -- Nom de la nouvelle collection.
  IN p_nom VARCHAR(25),
  -- Prix HT de la nouvelle collection.
  IN p_prix_ht DECIMAL(5,2),
  -- Identifiant de la nouvelle collection.
  OUT p_id INT
  )
BEGIN
  /*
  ** Insérer la nouvelle collection et
  ** récupérer l'identifiant affecté.
  */
  INSERT INTO collection (nom,prix_ht)
  VALUES (p_nom,p_prix_ht);
  SET p_id = LAST_INSERT_ID();
END;
```

*Script PHP (requête non préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Définition des caractéristiques de la nouvelle collection.
$nom = 'Objectif Solutions';
$prix_ht = 19.86;
// Exécution de la requête d'appel de la procédure.
// Le paramètre OUT de la procédure est récupéré dans la
// variable MySQL @id.
$sql = "CALL ps_creer_collection('$nom',$prix_ht,@id)";
$requête = mysqli_query($db,$sql);
// Exécution de la requête qui lit le contenu de la
// variable MySQL @id.
$sql='SELECT @id';
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
// Affichage du résultat.
echo 'Identifiant de la nouvelle collection = ',
     $ligne['@id'],'<br />';
// Exécution d'une requête qui supprime la nouvelle
// collection.
$sql = "DELETE FROM collection WHERE id = {$ligne['@id']}";
$requête = mysqli_query($db,$sql);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Identifiant de la nouvelle collection = 9
```

*Script PHP (requête préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Définition des caractéristiques de la nouvelle collection.
$nom = 'Objectif Solutions';
$prix_ht = 19.86;
// Exécution de la requête d'appel de la procédure.
// Le paramètre OUT de la procédure est récupéré dans la
// variable MySQL @id.
$sql = 'CALL ps_creer_collection(?,?,@id)';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'sd',$nom,$prix_ht);
$ok = mysqli_stmt_execute($requête);
// Exécution de la requête qui lit le contenu de la
// variable MySQL @id.
$sql='SELECT @id';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_result($requête,$id);
$ok = mysqli_stmt_execute($requête);
$ok = mysqli_stmt_fetch($requête);
// Affichage du résultat.
echo "Identifiant de la nouvelle collection = $id";
// Exécution d'une requête qui supprime la nouvelle
// collection.
$sql = "DELETE FROM collection WHERE id = $id";
$requête = mysqli_query($db,$sql);
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Identifiant de la nouvelle collection = 10
```

## Deuxième exemple : procédure qui retourne un résultat directement

Dans ce deuxième exemple, nous appellerons une procédure stockée qui retourne la liste des sous-rubriques d'une rubrique dont l'identifiant est passé en paramètre.

*Code source de la procédure stockée*

```
CREATE PROCEDURE ps_lire_sous_rubriques
  (
  -- Identifiant d'une rubrique (parent).
  IN p_id_parent INT
  )
BEGIN
  /*
  ** Sélectionner les sous-rubriques d'une
  ** rubrique dont l'identifiant est passé
  ** en paramètre.
  */
  SELECT
    titre
  FROM
    rubrique
  WHERE
    id_parent = p_id_parent;
END;
```

*Script PHP (requête non préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Identifiant de la rubrique parent.
$id_rubrique = 1;
// Exécution de la requête d'appel de la procédure.
$sql = "CALL ps_lire_sous_rubriques($id_rubrique)";
$requête = mysqli_query($db,$sql);
while ($ligne = mysqli_fetch_assoc($requête)) {
  echo $ligne['titre'],'<br />';
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
MySQL
Oracle
```

*Script PHP (requête préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Identifiant de la rubrique parent.
$id_rubrique = 1;
// Exécution de la requête d'appel de la procédure.
$sql = 'CALL ps_lire_sous_rubriques(?)';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'i',$id_rubrique);
$ok = mysqli_stmt_execute($requête);
// Important de faire le bind du résultat après le execute
// car la structure du résultat n'est pas connue avant.
$ok = mysqli_stmt_bind_result($requête,$titre);
while (mysqli_stmt_fetch($requête)) {
  echo $titre,'<br />';
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
MySQL
Oracle
```

### b. Fonction stockée

Pour appeler une fonction stockée, il faut exécuter l'ordre SQL `SELECT nom_fonction(...)` à l'aide des fonctions `mysqli_query` ou `mysqli_stmt_execute` puis lire le résultat à l'aide des fonctions de fetch adaptées.

Dans l'exemple, nous appellerons une fonction stockée qui retourne le nombre de sous-rubriques d'une rubrique dont l'identifiant est passé en paramètre.

*Code source de la procédure stockée*

```
CREATE FUNCTION fs_nombre_sous_rubriques
  (
  -- Identifiant d'une rubrique (parent).
  p_id_parent INT
  )
  RETURNS INT
BEGIN
  /*
  ** Compter le nombre de sous-rubriques d'une
  ** rubrique dont l'identifiant est passé
  ** en paramètre.
  */
  DECLARE v_resultat INT;
  SELECT
    COUNT(*)
  INTO
    v_resultat
  FROM
    rubrique
  WHERE
    id_parent = p_id_parent;
  RETURN v_resultat;
END;
```

*Script PHP (requête non préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Identifiant de la rubrique parent.
$id_rubrique = 1;
// Exécution de la requête qui appelle la fonction
// (l'expression qui appelle la fonction est nommée avec
// un alias de colonne).
$sql = "SELECT fs_nombre_sous_rubriques($id_rubrique) nb";
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
echo 'Nombre de sous-rubriques = ',$ligne['nb'];
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Nombre de sous-rubriques = 2
```

*Script PHP (requête préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Identifiant de la rubrique parent.
$id_rubrique = 1;
// Exécution de la requête qui appelle la fonction
// (l'expression qui appelle la fonction est nommée avec
// un alias de colonne).
```

```
$sql = 'SELECT fs_nombre_sous_rubriques(?) nb';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'i',$id_rubrique);
$ok = mysqli_stmt_execute($requête);
$ok = mysqli_stmt_bind_result($requête,$nb);
$ok = mysqli_stmt_fetch($requête);
echo 'Nombre de sous-rubriques = ',$nb;
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
Nombre de sous-rubriques = 2
```

## 7. Utiliser les types de données BLOB

Lire ou mettre à jour (insérer, modifier) des données de type BLOB ne pose pas de problème particulier.

### a. Insertion ou modification

Dans le cas de l'insertion ou de la modification à l'aide d'une requête non préparée, il faut juste s'assurer que les caractères spéciaux de la donnée BLOB sont correctement protégés, par exemple en lui appliquant la fonction `mysqli_real_escape_string`. (cf. dans ce chapitre - "Magic quotes" : le retour).

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Titre du livre dont il faut charger l'image de couverture
// et nom du fichier image correspondant.
$titre= 'PHP 5.2';
$fichier_image = 'RI52PHP.jpg';
// Lecture du contenu du fichier image en s'assurant qu'il
// n'y a pas d'encodage "magic quotes".
set_magic_quotes_runtime(0);
$image = file_get_contents($fichier_image);
// Protection des caractères spéciaux de l'image.
$image = mysqli_real_escape_string($db,$image);
// Exécution d'une requête de mise à jour de l'image.
$sql = "UPDATE livre SET couverture = '$image' " .
       "WHERE titre = '$titre'";
$ok = mysqli_query($db,$sql);
if ($ok) {
  echo 'Mise à jour terminée avec succès.';
} else {
  echo 'Erreur lors de la mise à jour.',
       mysqli_error($db);
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat (si tout se passe bien)*

```
Mise à jour terminée avec succès.
```

L'opération de protection des caractères spéciaux n'est pas nécessaire lors de l'utilisation d'une requête préparée. Par contre, la donnée BLOB doit être "envoyée" au serveur MySQL à l'aide de la fonction `mysqli_stmt_send_long_data`.

*Syntaxe*

```
booléen mysqli_stmt_send_long_data(objet requête, entier
numéro_paramètre, chaîne donnée)
```

`requête`

Ressource de requête préparée retournée par la fonction `mysqli_prepare` (ou `mysqli_stmt_init`).

`numéro_paramètre`

Numéro du paramètre auquel les données sont associées (0 pour le premier paramètre).

`donnée`

Chaîne de caractères contenant les données à envoyer.

La fonction `mysqli_stmt_send_long_data` retourne `TRUE` en cas de succès et `FALSE` en cas d'erreur.

Si la longueur de la donnée à envoyer excède la limite `max_allowed_packet` de MySQL, cette fonction peut être appelée plusieurs fois pour envoyer les données par paquet de taille inférieure à la limite.

Par ailleurs, dans l'appel à la fonction `mysqli_stmt_bind_param`, il faut utiliser la lettre `b` (binaire) pour spécifier le type de la donnée.

*Exemple*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Titre du livre dont il faut charger l'image de couverture
// et nom du fichier image correspondant.
$titre= 'PHP 5.2';
$fichier_image = 'RI52PHP.jpg';
// Lecture du contenu du fichier image en s'assurant qu'il
// n'y a pas d'encodage "magic quotes".
set_magic_quotes_runtime(0);
$image = file_get_contents($fichier_image);
// Préparation de la requête de mise à jour de l'image.
$sql = 'UPDATE livre SET couverture = ? ' .
       'WHERE titre = ?';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'bs',$image,$titre);
// Envoi des données de l'image.
$ok = mysqli_stmt_send_long_data($requête,0,$image);
// Exécution de la requête.
$ok = mysqli_stmt_execute($requête);
if ($ok) {
  echo 'Mise à jour terminée avec succès.';
} else {
  echo 'Erreur lors de la mise à jour.',
       mysqli_stmt_error($requête);
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

### b. Lecture

La lecture d'une donnée de type `BLOB` ne nécessite pas d'opération particulière, à l'exception, éventuellement, de l'ajustement de la directive de configuration PHP `memory_limit`.

La directive `memory_limit` détermine la quantité de mémoire maximum qu'un script est autorisé à utiliser. La valeur par défaut dépend de la version de PHP (128 Mo en version 5.2.4, 8 Mo avant la version 5.2.0).

Cette directive peut être modifiée temporairement pour un script à l'aide de la fonction `ini_set`.

*Exemple*

```
ini_set('memory_limit','32M');
```

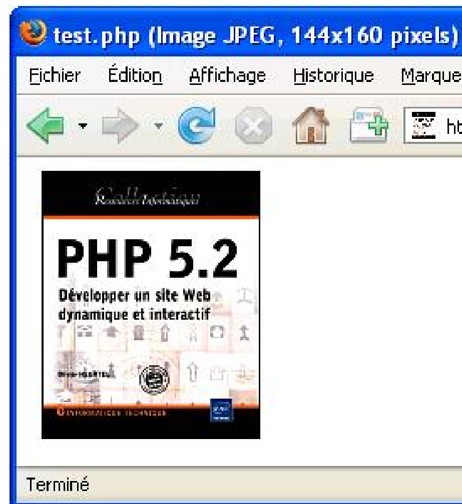
Dans le cas de l'utilisation d'une requête préparée, lors de l'appel à la fonction `mysqli_stmt_bind_result`, il semble que PHP alloue une variable pour la donnée BLOB d'une taille égale à la taille maximum possible du type (par exemple 16 Mo pour une données de type `MEDIUMBLOB`). Si cette taille est supérieure à la valeur de la directive `memory_limit`, une erreur se produit (même si la donnée à lire est en fait inférieure à la limite) : **`Fatal error`**`: Allowed memory size of 8388608 bytes exhausted (tried to allocate 16777216 bytes) ....`

*Exemple (requête non préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
```

```
// Titre du livre dont il faut afficher l'image
// de couverture et type MIME de l'image (ici JPEG).
$titre= 'PHP 5.2';
$type_mime = 'image/jpeg';
// Exécution d'une requête de lecture de l'image.
$sql = "SELECT couverture FROM livre WHERE titre = '$titre'";
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
// Déconnexion.
$ok = mysqli_close($db);
// Envoi de l'image au navigateur (le type MIME de l'image
// est communiqué au navigateur à l'aide de la fonction
// header).
header("Content-type: $type_mime");
echo $ligne['couverture'];
?>
```

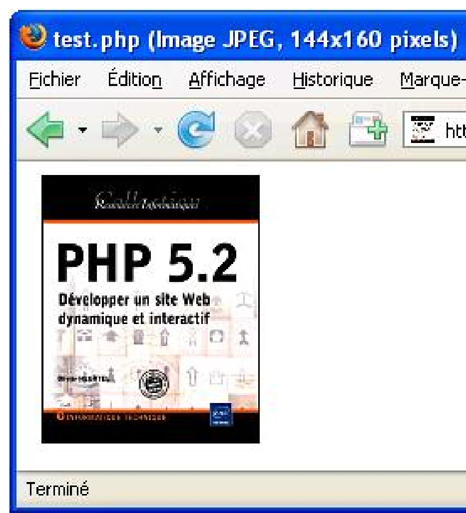
*Résultat*




*Exemple (requête préparée)*

```
<?php
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Titre du livre dont il faut afficher l'image
// de couverture et type MIME de l'image (ici JPEG).
$titre= 'PHP 5.2';
$type_mime = 'image/jpeg';
// Par précaution, augmentation de la valeur de la
// directive memory_limit.
ini_set('memory_limit','32M');
// Exécution d'une requête de lecture de l'image.
$sql = 'SELECT couverture FROM livre WHERE titre = ?';
$requête = mysqli_query($db,$sql);
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'s',$titre);
$ok = mysqli_stmt_bind_result($requête,$image);
$ok = mysqli_stmt_execute($requête);
$ok = mysqli_stmt_fetch($requête);
// Déconnexion.
$ok = mysqli_close($db);
// Envoi de l'image au navigateur (le type MIME de l'image
// est communiqué au navigateur à l'aide de la fonction
// header).
header("Content-type: $type_mime");
echo $image;
?>
```

*Résultat*


test.php (Image JPEG, 144x160 pixels)
Fichier
Édition
Affichage
Historique
Marque-
PHP 5.2
Développer un site Web
dynamique et interactif
Terminé

# Utilisation de l'extension MySQL

Toutes les fonctions présentées précédemment dans l'extension MySQLi, à l'exception des fonctions relatives aux requêtes préparées, ont un équivalent direct dans l'extension MySQL ; il suffit de remplacer le préfixe `mysqli_` par `mysql_` :

| MySQLi | MySQL | Syntaxe |
|---|---|---|
| `mysqli_connect` | `mysql_connect` | Voir remarque 1 |
| `mysqli_close` | `mysql_close` | Voir remarque 2 |
| `mysqli_select_db` | `mysql_select_db` | Voir remarque 3 |
| `mysqli_query` | `mysql_query` | Voir remarque 3 |
| `mysqli_num_rows` | `mysql_num_rows` | Identique |
| `mysqli_fetch_array` | `mysql_fetch_array` | Identique |
| `mysqli_fetch_array` | `mysql_fetch_array` | Identique |
| `mysqli_fetch_object` | `mysql_fetch_object` | Identique |
| `mysqli_fetch_row` | `mysql_fetch_row` | Identique |
| `mysqli_affected_rows` | `mysql_affected_rows` | Voir remarque 2 |
| `mysqli_insert_id` | `mysql_insert_id` | Voir remarque 2 |
| `mysqli_errno` | `mysql_errno` | Voir remarque 2 |
| `mysqli_error` | `mysql_error` | Voir remarque 2 |

*Remarques*

1

Les premiers paramètres de `mysql_connect` et `mysqli_connect` sont identiques (serveur, utilisateur, mot de passe) ; les paramètres suivants sont différents. La fonction `mysql_connect` ne permet pas de sélectionner la base de données à utiliser dès la connexion.

2

L'extension MySQL permet d'utiliser une connexion par défaut dans les appels de fonctions (la dernière connexion ouverte). En conséquence, l'identifiant de connexion est optionnel dans les fonctions `mysql_close`, `mysql_affected_rows`, `mysql_insert_id`, `mysql_errno` et `mysql_error`.

3

Pour les fonctions `mysql_select_db` et `mysql_query` la syntaxe est inversée : l'identifiant de connexion est passé en deuxième paramètre, et il devient facultatif (utilisation de la dernière connexion ouverte en cas d'omission).

Les fonctions `mysql_connect` et `mysql_query` retournent des ressources et non des objets. Dans la pratique, cela n'a aucune incidence car le résultat de ces fonctions est simplement passé en paramètre à d'autres fonctions.

L'extension MySQL n'offre pas de fonctions équivalentes aux fonctions `mysqli_connect_error` et `mysqli_connect_errno` qui permettent de récupérer des informations sur une éventuelle erreur de connexion. Par contre, les fonctions `mysql_error` et `mysql_errno` peuvent être appelées pour récupérer des informations sur une éventuelle erreur de connexion.

L'extension MySQL offre par ailleurs une fonction `mysql_`**`unbuffered`**`_query` qui propose la même syntaxe que la fonction `mysql_query`, et permet d'exécuter une requête mais sans placer le résultat dans un buffer. Le fonctionnement est

identique à l'appel de mysqli_query en passant la constante MYSQLI_USE_RESULT comme troisième paramètre.

*Exemple*

```
<?php
// Définir quelques variables.
$serveur = 'localhost';
$utilisateur = 'eniweb';
$mot_de_passe = 'web';
$base = 'eni';
// Connexion = mysql_connect
// Pas de sélection de la base dans cet appel.
$db = mysql_connect($serveur,$utilisateur,$mot_de_passe);
// Vérifier le succès de la connexion
if (mysql_errno() != 0) {
  printf(
    'Erreur mysql_connect : %d - %s<br />',
    mysql_errno(),mysql_error());
    exit;
}
// Sélectionner la base = mysql_select_db
// Différences par rapport à mysqli_select_db :
//   > syntaxe inversée
//   > identifiant de connexion optionnel
$ok = mysql_select_db($base,$db);
if (! $ok) {
  printf(
    'Erreur mysql_connect : %d - %s<br />',
    mysql_errno(),mysql_error());
    exit;
}
// Exécuter une requête = mysql_query
// Différences par rapport à mysqli_query :
//   > syntaxe inversée
//   > identifiant de connexion optionnel
//   > pas de troisième paramètre
$sql = 'SELECT * FROM collection LIMIT 4';
$requête = mysql_query($sql, $db);
if (! $requête) {
  printf(
    'Erreur mysql_connect : %d - %s<br />',
    mysql_errno(),mysql_error());
    exit;
}
// Déterminer le nombre de lignes du résultat = mysql_num_rows
// Identique à mysqli_num_rows.
printf("Nombre de collections = %s<br />\n",
       mysql_num_rows($requête));
// Récupérer le résultat = mysql_fetch_*
// Différences par rapport à mysqli_fetch_* :
//   > constantes MYSQL_NUM, MYSQL_ASSOC, MYSQL_BOTH
//     dans mysql_fetch_array
$ligne = mysql_fetch_array($requête, MYSQL_ASSOC);
printf(
  "Première collection = %s - %s<br />\n",
  $ligne['nom'],$ligne['prix_ht']);
// Exécuter une requête = mysql_query
// Exemple avec INSERT.
$sql = "INSERT INTO collection(nom,prix_ht) " .
       "VALUES('Open IT',5)";
$requête = mysql_query($sql, $db);
// Récupérer l'identifiant généré = mysql_insert_id
// Différences par rapport à mysqli_insert_id :
//   > identifiant de connexion optionnel
printf(
  "Identifiant de la nouvelle collection = %s<br />\n",
  mysql_insert_id($db));
// Exécuter une requête = mysql_query
// Exemple avec UPDATE.
```

```
$sql = "UPDATE collection SET prix_ht = 5.69 " .
       "WHERE nom = 'Open IT'";
$requête = mysql_query($sql, $db);
// Récupérer le nombre de lignes mises à jour =
// mysql_affected_rows
// Différences par rapport à mysqli_affected_rows :
//   > identifiant de connexion optionnel
printf(
  "Nombre de collection(s) modifiée(s) = %s<br />\n",
  mysql_affected_rows($db));
// Déconnexion = mysql_close
// Différences par rapport à mysqli_close :
//   > identifiant de connexion optionnel
mysql_close($db);
?>
```

*Résultat*

```
Nombre de collections = 4
Première collection = Ressources Informatiques - 25.66
Identifiant de la nouvelle collection = 11
Nombre de collection(s) modifiée(s) = 1
```

# PHP Data Objects (PDO)

*PHP Data Objects* (PDO) est une extension apparue en version 5.1 qui définit une interface uniforme pour accéder aux bases de données en PHP. L'accès à une base de données à travers PDO s'effectue par l'intermédiaire d'un driver qui expose les fonctionnalités de la base de données.

Il faut bien noter que PDO ne fournit pas une couche d'abstraction de la base de données mais une couche d'abstraction de l'accès aux bases de données. Les requêtes que vous écrivez doivent respecter la syntaxe de la base de données que vous utilisez ; PDO ne récrit pas les requêtes SQL et n'émule pas les fonctionnalités manquantes (à l'exception des requêtes paramétrées).

De nombreuses bases de données disposent d'un driver PDO parmi lesquelles MySQL, Oracle, Microsoft SQL Server et SQLite.

PDO est une extension orientée objet qui propose 3 classes :

- `PDO` : connexion entre PHP et la base de données,
- `PDOStatement` : requête préparée, et, après exécution, résultat associé,
- `PDOException` : exception levée par PDO.

Dans ce chapitre, nous présenterons cette extension par l'intermédaire d'un simple exemple commenté :

```
<?php
// Définition des paramètres de connexion.
// La syntaxe de la source (Data Source Name ou DSN)
// est spécifique à chaque driver.
$source = 'mysql:host=localhost;dbname=eni';
$utilisateur = 'eniweb';
$mot_de_passe = 'web';
// Définition de deux requêtes de test.
// Noter que la requête d'insertion est paramétrée (c'est
// en fait la seule fonctionnalité qui est émulée par PDO,
// si elle n'est pas nativement supportée par la base de
// données).
$sql_select = 'SELECT * FROM collection';
$sql_insert = 'INSERT INTO collection(nom,prix_ht) ' .
              'VALUES(:p1,:p2)';
// Toutes les opérations sont effectuées dans un bloc
// 'try' afin de récupérer les exceptions levées par PDO.
try {
  // Connexion à la base de données.
  $db = new PDO($source, $utilisateur, $mot_de_passe);
  // Modification des paramètres de la connexion pour
  // demander que des exceptions soient levées en cas
  // d'erreur.
  $db->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
  // Préparer une requête pour l'insertion.
  $st = $db->prepare($sql_insert);
  // Lier les paramètres.
  $st->bindParam(':p1',$nom);
  $st->bindParam(':p2',$prix_ht);
  // Affecter une valeur aux variables.
  $nom = 'Mega +';
  $prix_ht = 21.38;
  // Exécuter la requête.
  // Pour les bases de données qui supportent les transactions,
  // les méthodes beginTransaction(), commit() et rollback()
  // des objets PDO peuvent être utilisées.
  $st->execute();
  // Préparer une requête pour la sélection.
  $st = $db->prepare($sql_select);
  // Exécuter la requête.
  $st->execute();
  // Récupérer le résultat.
  // Plusieurs méthodes sont disponibles pour récupérer
```

```
  // le résultat : fetch(), fetchObject(), fetchAll().
  // La méthode fetch() dispose d'un paramètre qui permet
  // de spécifier le type de résultat (tableau, objet, etc.).
  while ($ligne = $st->fetch()) {
    echo "$ligne[1] - $ligne[2]<br />\n";
  }
  // Libérer les resources.
  $st = null;
  $db = null;
} catch (PDOException $e) {
  // Gérer les exceptions
  echo 'Error!: ',$e->getMessage(),'<br />';
  die();
}
?>
```

*Résultat*

```
Ressources Informatiques - 25.66
TechNote - 9.95
Les TP Informatiques - 25.59
Coffret Technique - 46.45
Coffret Solutions - 55.92
Epsilon - 51.18
Expert IT - 36.97
Solution Informatiques - 20.00
Objectif Solutions - 19.86
Open IT - 5.69
Mega + - 21.38
```

# "Magic quotes" : le retour

## 1. Préambule

Nous avons vu dans le chapitre Utiliser les fonctions PHP que PHP propose une fonctionnalité, appelée "magic quotes", dont l'objectif principal est de résoudre un problème lié à l'enregistrement des données dans une base de données, en effectuant un encodage sur les données saisies dans un formulaire.

*Exemple (insertion dans une base de données MySQL d'une donnée qui contient une apostrophe)*

```
<?php
// Donnée qui pose problème (peut être saisie innocemment
// dans un formulaire).
$nom = "L'Atout Réussite";
$prix_ht = 10;
// Requête d'insertion.
$sql = "INSERT INTO collection(nom,prix_ht) " .
       "VALUES('$nom',$prix_ht)";
echo $sql,'<br />';
// Exécution.
$db = mysqli_connect('localhost','eniweb','web','eni');
$requête = mysqli_query($db,$sql);
echo mysqli_error($db),'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
INSERT INTO collection(nom,prix_ht) VALUES('L'Atout Réussite',10)
You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax
to use near 'Atout Réussite',10)' at line 1
```

En SQL, le délimiteur de chaîne de caractères est l'apostrophe : si une requête envoie la chaîne `'L'Atout Réussite'` à la base, cette dernière va interpréter `'L'` comme une chaîne et ne saura pas quoi faire du reste (`Atout Réussite'`).

Pour régler ce problème, il faut indiquer à la base que les apostrophes à l'intérieur de la chaîne ne sont pas les délimiteurs de la chaîne, généralement en faisant précéder l'apostrophe d'un caractère "magique" ("d'échappement") : c'est le caractère anti-slash (\) ou apostrophe (') pour MySQL.

*Exemple*

```
<?php
// Donnée corrigée.
$nom = "L\'Atout Réussite";
$prix_ht = 10;
// Requête d'insertion.
$sql = "INSERT INTO collection(nom,prix_ht) " .
       "VALUES('$nom',$prix_ht)";
echo $sql,'<br />';
// Exécution.
$db = mysqli_connect('localhost','eniweb','web','eni');
$requête = mysqli_query($db,$sql);
echo mysqli_error($db),'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
INSERT INTO collection(nom,prix_ht) VALUES('L\'Atout Réussite',10)
```

La fonctionnalité "magic quotes" d'encodage automatique répond à cette problématique : si elle est active (directive de configuration `magic_quotes_gpc = on`), toutes les données issues d'un formulaire (méthodes `GET` ou `POST`), d'une URL (méthode `GET`) ou d'un cookie sont automatiquement encodées avec le caractère anti-slash (\), ou apostrophe (') si la

directive de configuration magic_quotes_sybase est à on.

Malheureusement, nous avons vu que cette fonctionnalité "magic quotes" posait d'autres problèmes vis à vis :

- de l'affichage des données dans la page HTML ;
- de la dépendance potentielle du code à des directives de configuration.

Pour mémoire, la solution proposée consiste à s'assurer que les données sont chargées dans des variables sans encodage et à faire ce qu'il faut au moment de l'enregistrement dans la base ; c'est ce que nous allons voir dans ce chapitre.

## 2. Chargement des données en provenance d'une base

Comme nous l'avons vu dans le chapitre Utiliser les fonctions PHP, lorsque la directive magic_quotes_runtime est à on, une protection "magic quotes" est automatiquement appliquée aux données lues dans une base de données MySQL. En complément, la fonction get_magic_quotes_runtime permet de connaître la valeur de la directive magic_quotes_runtime et la fonction set_magic_quotes_runtime de la modifier en cours de script.

*Exemple*

```
<?php
// Requête de test.
$sql = "SELECT 'L''Atout Réussite' data";
// Connexion.
$db = mysqli_connect('localhost','eniweb','web','eni');
// Affichage de la valeur de la directive magic_quotes_runtime.
$mqr = get_magic_quotes_runtime();
echo "<b>magic_quotes_runtime = $mqr</b><br />";
// Exécution de la requête et affichage du résultat.
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
echo $ligne['data'],'<br />';
// Modification de la valeur de la directive
// magic_quotes_runtime.
$mqr = ($mqr == 1)?0:1;
set_magic_quotes_runtime($mqr);
echo "<b>magic_quotes_runtime = $mqr</b><br />";
// Exécution de la requête et affichage du résultat.
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
echo $ligne['data'],'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
magic_quotes_runtime = 0
L'Atout Réussite
magic_quotes_runtime = 1
L\'Atout Réussite
```

La fonction set_magic_quotes_runtime est particulièrement intéressante avec MySQL pour écrire un code indépendant de la configuration : avant chaque exécution d'une requête SELECT, il suffit d'appeler cette fonction avec la valeur 0 ou 1 correspondant à la stratégie que vous avez adoptée .

Dans le même ordre d'idée, un appel du type set_magic_quotes_runtime(get_magic_quotes_gpc()) permet d'obtenir, pour les données issues de la base, le même comportement que pour les données GPC.

## 3. Mise à jour des données dans la base

Pour la mise à jour des données dans la base, il faut s'assurer que toutes les données de type "texte" ont le caractère d'échappement adapté (\ ou ' pour MySQL) devant chaque apostrophe.

Si vous avez adopté une stratégie dans laquelle toutes les variables contiennent des données encodées, il n'y a rien à faire.

À l'inverse, si vous avez adopté une stratégie (recommandée dans cet ouvrage) dans laquelle toutes les variables ne contiennent pas de données encodées, il convient d'assurer l'échappement des apostrophes dans les données envoyées à la base.

La fonction `mysqli_real_escape_string` peut être utilisée pour cela.

*Syntaxe*

```
chaîne mysqli_real_escape_string(objet connexion,chaîne valeur)
```

*Avec*

`connexion`

Identifiant de connexion retourné par la fonction `mysqli_connect`.

`valeur`

Chaîne de caractères à protéger.

La fonction `mysqli_real_escape_string` ajoute un anti-slash (\) devant tous les caractères apostrophe ('), guillemet ("), anti-slash (\), NUL (ASCII 0), retour à la ligne (`\n`) et retour chariot (`\r`) trouvés dans la chaîne `valeur`.

---

Le paramètre `valeur` ne doit pas déjà être protégé.

---

*Exemple*

```
<?php
// Donnée qui pose problème (peut être saisie innocemment
// dans un formulaire).
$nom = "L'Atout Réussite";
$prix_ht = 10;
// Connexion.
$db = mysqli_connect('localhost','eniweb','web','eni');
// Définition de la requête avec protection des données
// de type "texte".
$sql =
  sprintf(
    "INSERT INTO collection(nom,prix_ht) VALUES('%s',%s)",
    mysqli_escape_string($db,$nom),$prix_ht);
echo $sql,'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
INSERT INTO collection(nom,prix_ht) VALUES('L\'Atout Réussite',10)
```

Il est possible d'écrire une fonction générique, dans le même esprit que la fonction `sprintf`, en marquant l'emplacement des paramètres par une séquence `%n`, `n` valant `1` pour le 1er paramètre, `2` pour le deuxième... Cette fonction accepte un nombre variable de paramètres, le premier étant la connexion à utiliser, le deuxième la structure de la requête et les suivants les valeurs des paramètres dans l'ordre de numérotation.

*Exemple*

```
<?php
function construire_requête($db,$sql) {
  // Récupérer le nombre de paramètres.
  $nombre_param = func_num_args();
  // Boucler sur tous les paramètres à partir du troisième.
  for($i=2;$i<$nombre_param;$i++) {
    // Récupérer la valeur du paramètre.
    $valeur = func_get_arg($i);
    // Si c'est une chaîne, l'échapper.
```

```
    if (is_string($valeur)) {
      $valeur = mysqli_escape_string($db,$valeur);
    }
    // Mettre la valeur à son emplacement %n (n = $i-1).
    $sql = str_replace('%'.($i-1),$valeur,$sql);
  }
  // Retourner la requête.
  return $sql;
}
// Des variables contiennent des valeurs venant
// de quelque part ...
$nom = "L'Atout Réussite";
$prix_ht = 10;
// Connexion.
$db = mysqli_connect('localhost','eniweb','web','eni');
// Construction de la requête.
$sql =
  construire_requête(
    $db,
    " INSERT INTO collection(nom,prix_ht) VALUES('%1',%2)",
    $nom,
    $prix_ht);
echo $sql,'<br />';
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
INSERT INTO collection(nom,prix_ht) VALUES('L\'Atout Réussite',10)
```

 Il n'y a pas de problème de ce type lors de l'utilisation de requêtes paramétrées.

*Exemple*

```
<?php
// Des variables contiennent des valeurs venant
// de quelque part ...
$nom = "L'Atout Réussite +";
$prix_ht = 10;
// Connexion.
$db = mysqli_connect('localhost','eniweb','web','eni');
// Execution d'une requête d'insertion.
$sql = 'INSERT INTO collection(nom,prix_ht) VALUES(?,?)';
$requête = mysqli_prepare($db,$sql);
$ok = mysqli_stmt_bind_param($requête,'sd',$nom,$prix_ht);
$ok = mysqli_execute($requête);
if ($ok) {
  echo mysqli_stmt_affected_rows($requête),
      ' collection insérée<br />';
} else {
  echo mysqli_stmt_error($requête),'<br />';
}
// Déconnexion.
$ok = mysqli_close($db);
?>
```

*Résultat*

```
1 collection insérée.
```

# Exemples d’intégration dans des formulaires

## 1. Vue d’ensemble

Pour terminer ce chapitre, nous allons présenter quelques exemples d’accès à une base de données MySQL à partir de formulaires.

Trois exemples sont proposés :

- un formulaire qui permet de saisir des données dans une liste ;
- un formulaire de recherche avec affichage du résultat ;
- un formulaire de saisie de type "page".

Pour des raisons de concision, dans ces exemples, le contrôle de la saisie et la gestion d’erreur sont pratiquement absents, et la mise en forme est très simple.

## 2. Formulaire de saisie en liste

*Présentation du formulaire*

| Identifiant | Nom | Prix H.T. | Supprimer |
|---|---|---|---|
| 1 | Ressources Informatiqu | 24,44 | ☐ |
| 2 | TechNote | 9,48 | ☐ |
| 3 | Les TP Informatiques | 25,59 | ☐ |
| 4 | Coffret Technique | 46,45 | ☐ |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Enregistrer

Le formulaire propose le contenu actuel de la table qui peut être modifié (saisie directe dans les zones) ou supprimé (par les cases à cocher), plus cinq lignes vides qui permettent de saisir de nouvelles valeurs. Dans tous les cas, l’identifiant ne peut pas être saisi, c’est le serveur MySQL qui va l’attribuer.

Chaque ligne du tableau contient 4 zones de formulaire qui sont nommées (attribut `name` de la balise `<input>`) de la manière suivante :

| Colonne | Nom |
|---|---|
| Identifiant | `saisie[i][modifier]` |
| Nom | `saisie[i][nom]` |
| Prix H.T. | `saisie[i][prix_ht]` |

| | |
|---|---|
| Supprimer | saisie[i][supprimer] |

L'indice i est l'identifiant de la collection pour les lignes qui existent et un numéro compris entre -1 et -5 pour les lignes vides. La zone de la colonne **Identifiant** est une zone masquée (type="hidden") qui va être employée pour identifier les lignes dans lesquelles l'utilisateur a effectué une modification.

Avec ce processus de nommage, toute la saisie est récupérée dans le script PHP sous la forme d'un tableau multidimensionnel. Chaque ligne du tableau correspond à une ligne du formulaire avec la clé égal à l'identifiant (ou -1 à -5 pour les nouvelles lignes) et la valeur égale à un tableau associatif donnant les éléments saisis.

Pour identifier les lignes modifiées par l'utilisateur, les zones de saisie du nom et du prix des lignes existantes contiennent le code JavaScript suivant :

```
onChange="document.formulaire[$n].value=1"
```

Ce petit bout de code JavaScript a pour effet, à chaque fois que la zone en question est modifiée, de mettre un 1 dans la zone masquée associée à la ligne. Le formulaire s'appelant formulaire (<form name = "formulaire"...), l'expression document.formulaire[n] désigne la nième zone du formulaire formulaire du document courant, la première zone du formulaire ayant le numéro 0. Dans le code source, la variable $n est calculée pour chaque ligne $i du formulaire par la formule $n = 4 * ($i - 1) : la zone cachée de la ligne 1 a le numéro 0 (c'est la première du formulaire), celle de la ligne 2 le numéro 4 et ainsi de suite.

Cet exemple peut (doit) être amélioré :

- pour contrôler la saisie de l'utilisateur ;
- pour gérer les erreurs.

*Source*

```
<?php
// Inclusion du fichier qui contient les fonctions générales.
include('fonctions.inc');
// Connexion.
$db = mysqli_connect('localhost','eniweb','web','eni');
// Traitement du formulaire.
if (isset($_POST['ok'])) {
  // Supprimer la protection "magic quotes" éventuelle
  // des tableaux GPC.
  supprimer_encodage_MQ_GPC();
  // Récupérer le tableau contenant la saisie.
  $lignes = $_POST['saisie'];
  foreach($lignes as $id => $ligne) {
    // Nettoyage de la saisie.
    $nom = trim($ligne['nom']);
    // Pour le prix, remplacer la virgule par un point
    // et supprimer les espaces.
    $prix_ht = str_replace(',','.',$ligne['prix_ht']);
    $prix_ht = str_replace(' ','',$prix_ht);
    // A ce stade, il faudrait vérifier la saisie ...
    // Définition de la requête à exécuter.
    // Pour chaque action, nous allons utiliser une requête
    // préparée. Lorsqu'un cas est rencontré pour la première
    // fois, il faut préparer la requête et lier les variables.
    $requête = NULL;
    if ($id < 0 and $nom.$prix_ht != '') {
      // Identifiant négatif et quelque chose de saisi
      // = création = INSERT
      if (! $req_ins) {
        $sql = 'INSERT INTO collection(nom,prix_ht) ' .
               'VALUES(?,?)';
        $req_ins = mysqli_prepare($db,$sql);
        $ok = mysqli_stmt_bind_param
                ($req_ins,'sd',$nom,$prix_ht);
      }
      $requête = $req_ins;
    } elseif (isset($ligne['supprimer'])) {
      // Case "supprimer" cochée = suppression = DELETE
```

```
      if (! $req_del) {
        $sql = ’DELETE FROM collection WHERE id = ?’;
        $req_del = mysqli_prepare($db,$sql);
        $ok = mysqli_stmt_bind_param($req_del,’i’,$id);
      }
      $requête = $req_del;
    } elseif ($ligne[’modifier’]==1) {
      // Zone "modifier" TRUE (1) = modification = UPDATE
      if (! $req_upd) {
        $sql = ’UPDATE collection ’ .
               ’SET nom = ?, prix_ht = ? ’ .
               ’WHERE id = ?’;
        $req_upd = mysqli_prepare($db,$sql);
        $ok = mysqli_stmt_bind_param
                ($req_upd,’sdi’,$nom,$prix_ht,$id);
      }
      $requête = $req_upd;
    }
    // Si une requête a été définie, l’exécuter.
    if ($requête) {
      $ok = mysqli_execute($requête);
      // A ce stade, il faudrait tester les erreurs ...
    }
  }
}
// Recharger les collections (ici avec une requête
// non préparée).
$sql = ’SELECT * FROM collection’;
$résultat = mysqli_query($db,$sql);
// A ce stade, il faudrait tester les erreurs ...
// Affichage de la page ...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Gestion des collections</title></head>
  <body>
    <!-- construction d’une table HTML à l’intérieur
    ++++ d’un formulaire -->
    <form action="saisie-liste.php" name="formulaire" method="post">
    <table border="1" cellpadding="4" cellspacing="0">
    <!-- ligne de titre -->
    <tr align="center">
    <th>Identifiant</th><th>Nom</th><th>Prix H.T.</th>
                <th>Supprimer</th>
    </tr>
    <?php
    // Code PHP pour les lignes du tableau.
    if ($résultat) { // S’il y a un résultat à afficher
      // Initialisation d’un compteur de ligne.
      $i = 0;
      // Boucle de fetch.
      while ($ligne = mysqli_fetch_assoc($résultat)) {
        // Incrémentation du compteur de ligne.
        $i++;
        // Calcul du numéro d’ordre dans le formulaire de la
        // zone cachée correspondant à l’identifiant.
        $n = 4 * ($i - 1);
        // Mise en forme des données.
        $ligne[’nom’] = vers_page($ligne[’nom’]);
        $ligne[’prix_ht’] =
          vers_page
            (number_format($ligne[’prix_ht’],2,"," ," "));
        // Génération de la ligne de la table HTML.
        // Insertion des balises INPUT du formulaire.
        printf(
        "<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>",
        "$ligne[id]
         <input type=\"hidden\"
```

```
            name=\"saisie[$ligne[id]][modifier]\" />",
        "<input type=\"text\"
            name=\"saisie[$ligne[id]][nom]\"
            value=\"$ligne[nom]\"
            onchange=\"document.formulaire[$n].value=1\" />",
        "<input type=\"text\"
            name=\"saisie[$ligne[id]][prix_ht]\"
            value=\"$ligne[prix_ht]\"
            onchange=\"document.formulaire[$n].value=1\" />",
        "<input type=\"checkbox\"
            name=\"saisie[$ligne[id]][supprimer]\"
            value=\"$ligne[id]\" />");
      } // while
      // Ajout de 5 lignes vides pour la création
      // (sans identifiant, sans case de suppression).
      for($i=1;$i<=5;$i++) {
        printf(
        "<tr><td>%s</td><td>%s</td><td>%s</td><td>%s</td></tr>",
        "",
        "<input type=\"text\"
            name=\"saisie[-$i][nom]\"
            value=\"\" />",
        "<input type=\"text\"
            name=\"saisie[-$i][prix_ht]\"
            value=\"\" />",
        "");
      } // for
    }
    ?>
    </table>
    <p><input type="submit" name="ok" value="Enregistrer" /></p>
    </form>
  </body>
</html>
```

## 3. Formulaire de recherche

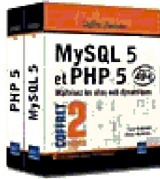
*Présentation du formulaire*

- Affichage initial :

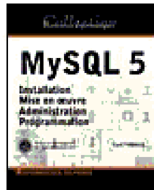
Rechercher : OK

- Résultat d'une recherche :

Rechercher : mysql OK




**MySQL 5 et PHP 5**
Maîtrisez les sites web dynamiques
Collection : Coffret Technique




**MySQL 5**
Installation, mise en œuvre, administration et programmation
Collection : Ressources Informatiques




**PHP 5**
L'accès aux données (MySQL, Oracle, SQL Server, SQLite...)
Collection : TechNote




**PHP et MySQL (versions 4 et 5)**
Entraînez-vous à créer des applications professionnelles
Collection : Les TP Informatiques

Le formulaire de recherche est très simple : il effectue une recherche en texte intégral sur un texte saisi dans l'unique zone. Le résultat est affiché sous la forme d'une liste avec l'image de couverture du livre et quelques informations sur le livre.

Pour que la recherche en texte intégral fonctionne, il faut créer un index `FULLTEXT` (cf. chapitre Techniques avancées avec MySQL - Effectuer des recherches en texte intégral).

*Exemple*

```
mysql> CREATE FULLTEXT INDEX ix_texte
    -> ON livre(titre,sous_titre,description);
Query OK, 9 rows affected (0.01 sec)
Records: 9  Duplicates: 0  Warnings: 0
```

*Source*

```
<?php
// Inclusion du fichier qui contient les fonctions générales.
include('fonctions.inc');
// Variable pour un éventuel message.
$message = '';
// Traitement du formulaire.
if (isset($_POST['ok'])) {
  // Supprimer la protection "magic quotes" éventuelle
  // des tableaux GPC.
  supprimer_encodage_MQ_GPC();
  // Récupérer le texte saisi.
  $recherche = $_POST['recherche'];
  if (empty($recherche)) {
    $message = 'Vous devez saisir le texte recherché.';
  } else {
    // Connexion.
    // Utilisation de l'opérateur @ pour masquer les alertes.
    $db = @mysqli_connect('localhost','eniweb','web','eni');
    if ($db === FALSE) {
      $message = 'Erreur de connexion à la base de données.';
```

```
    } else {
      // Exécuter la requête de recherche.
      $sql = 'SELECT liv.id,liv.titre,liv.sous_titre,col.nom ' .
             'FROM livre liv JOIN collection col ' .
             '     ON (liv.id_collection = col.id) ' .
             'WHERE MATCH(titre,sous_titre,description) ' .
             '      AGAINST(?)';
      if ($requête = mysqli_prepare($db,$sql)) {
        $ok = mysqli_stmt_bind_param($requête,'s',$recherche);
        if ($ok) {
          $ok = mysqli_stmt_bind_result
                ($requête,$id,$titre,$sous_titre,$collection);
        }
        if ($ok) { $ok = mysqli_execute($requête); }
        if (! $ok) { $message = mysqli_stmt_error($requête);}
      } else {
        $message = mysqli_error($db);
      }
      if ($message) {
        $message = "Erreur lors de la recherche ($message).";
      }
    }
  }
}
// Affichage de la page ...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Rechercher un livre</title></head>
  <body>
    <!-- Formulaire de recherche (très simple !) -->
    <form action="recherche-livre.php" method="post">
    <div>Rechercher :
    <input type="text" name="recherche"
       value="<?php echo vers_formulaire($recherche) ?>" />
    <input type="submit" name="ok" value="OK" /></div>
    </form>
    <!-- Résultat de la recherche -->
    <?php
    // Compteur du nombre de livres trouvés.
    $nombre_livres = 0;
    if ($requête) { // S'il y a un résultat à afficher
      // Balise d'ouverture de la table HTML.
      echo '<table border="0" cellpadding="4">',"\n";
      // Boucle de fetch.
      while (mysqli_stmt_fetch($requête)) {
        $nombre_livres++;
        // Mise en forme des données.
        $titre = vers_page($titre);
        $sous_titre = vers_page($sous_titre);
        $collection = vers_page($collection);
        // Génération de la ligne de la table HTML.
        // L'image est affichée par appel à un autre script.
        printf(
        "<tr><td>%s</td><td>%s<br />%s<br />%s</td></tr>\n",
        "<img alt=\"\" src=\"image-livre.php?id=$id\" />",
        "<b>$titre</b>",
        $sous_titre,
        "Collection : $collection");
      } // while
      // Balise de fermeture de la table HTML.
      echo '</table>',"\n";
      // Si le résultat est vide, afficher un message.
      if ($nombre_livres == 0) {
        $message = 'Aucun livre trouvé.';
      }
    } // if ($requête)
    ?>
```

```
    <div><?php echo vers_page($message); ?></div>
  </body>
</html>
```

L'image est affichée à l'aide d'un deuxième script.

*Source*

```
<?php
// Récupération de l'identifiant du livre dont il faut
// afficher l'image de couverture : passé dans l'URL.
// Utilisation d'un filtre pour s'assurer que la valeur
// passée est bien un entier.
$id = filter_input(INPUT_GET,'id',FILTER_VALIDATE_INT);
// Le filtre a "échoué" : quitter le script.
if ($id === FALSE OR $id === NULL) {
  exit('Paramètre invalide ou absent.');
}
// Connexion et sélection de la base de données.
$db = mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Echec de la connexion.');
}
// Type MIME des images.
$type_mime = 'image/jpeg';
// Exécution d'une requête de lecture de l'image.
$sql = "SELECT couverture FROM livre WHERE id = $id";
$requête = mysqli_query($db,$sql);
$ligne = mysqli_fetch_assoc($requête);
// Déconnexion.
$ok = mysqli_close($db);
// Envoi de l'image au navigateur (le type MIME de l'image
// est communiqué au navigateur à l'aide de la fonction
// header).
header("Content-type: $type_mime");
echo $ligne['couverture'];
?>
```

## 4. Formulaire de saisie

*Présentation du formulaire*

- Affichage initial :

Identifiant : Charger

- Livre affiché dans le formulaire :

Identifiant : 2 Charger

ISBN : 978-2-7460-3992-6
Titre : PHP 5.2
Sous-titre : Développer un site Web dynamique et interactif
Nombre pages : 518 Année de parution : 2007
Niveau : Débutant Initié Confirmé Expert
Collection : Ressources Informatiques
Rubriques :
Base de données - MySQL
Base de données - Oracle
Développement - Langages
Développement - Méthode
Internet - HTML - XML
Internet - Conception Web
Internet - Sécurité
Open Source - Système
Description :

Couverture : Parcourir...
Enregistrer

Le petit formulaire affiché en haut de la page permet de saisir l'identifiant du livre qu'il faut modifier ; lorque l'utilisateur clique sur le bouton **Charger**, le livre est chargé dans le formulaire.

Le bouton **Enregistrer** permet d'enregistrer les modifications dans la base. Une fois enregistré; le livre est rechargé dans le formulaire pour contrôle.

Le bouton **Parcourir** permet de sélectionner un fichier qui sera stocké dans la base et associé comme image de couverture au livre. Lorsqu'une image de couverture est associée au livre, celle-ci est affichée dans le formulaire, avec la même technique que dans le formulaire de recherche.

*Exemple*

PHP 5.2
Parcourir...
Enregistrer

*Source*

```
<?php
// Inclusion du fichier qui contient les fonctions générales.
include('fonctions.inc');
// Pas de protection "magic quotes" pour les données lues dans
// la base de données ou dans les fichiers.
set_magic_quotes_runtime(0);
// Suppression de la protection "magic quotes" éventuelle
// des tableaux GPC.
supprimer_encodage_MQ_GPC();
// Variables qui indiquent si un livre doit être chargé et/ou
```

```
// si un livre doit être enregistré.
$charger_livre = FALSE;
$enregistrer_livre = FALSE;
// Variable pour un éventuel message.
$message = '';
// Tester si le script est appelé en traitement d'un formulaire.
if (isset($_POST['charger']) OR isset($_POST['ok'])) { // oui
  // Récupérer l'identifiant du livre.
  // Utilisation d'un filtre pour s'assurer que la valeur
  // récupérée est bien un entier.
  $id_livre = filter_input(INPUT_POST,'id',FILTER_VALIDATE_INT);
  // Identifiant absent ou invalide => message.
  // Sinon, déterminer l'action à effectuer.
  if ($id_livre === FALSE OR $id_livre === NULL) {
    $message = 'Identifiant absent ou invalide.';
  } else {
    $enregistrer_livre = isset($_POST['ok']);
    $charger_livre = // enregistrer => rechargement
      (isset($_POST['charger']) OR $enregistrer_livre);
  }
}
// Connexion si nécessaire.
if ($charger_livre OR $enregistrer_livre) {
  // Utilisation de l'opérateur @ pour masquer les alertes.
  $db = @mysqli_connect('localhost','eniweb','web','eni');
  // En cas d'erreur, on arrête tout.
  if ($db === FALSE) {
    $message = 'Erreur de connexion à la base de données.';
    $charger_livre = FALSE;
    $enregistrer_livre = FALSE;
  }
}
// S'il y a un livre à enregistrer ...
if ($enregistrer_livre) {
  // Récupérer le contenu du formulaire.
  // Il faudrait vérifier la saisie ...
  $livre = $_POST;
  // Enregistrement du livre.
  $sql = 'UPDATE livre SET ' .
           'isbn = ?,' .
           'titre = ?,' .
           'sous_titre = ?,' .
           'nombre_pages = ?,' .
           'annee_parution = ?,' .
           'niveau = ?,' .
           'id_collection = ?,' .
           'description = ? ' .
         'WHERE id = ?';
  if ($ok = ($req_maj = mysqli_prepare($db,$sql))) {
    $ok = mysqli_stmt_bind_param
            (
            $req_maj,
            'sssiisisi',
            $livre['isbn'],
            $livre['titre'],
            $livre['sous_titre'],
            $livre['nombre_pages'],
            $livre['annee_parution'],
            $livre['niveau'],
            $livre['collection'],
            $livre['description'],
            $id_livre
            );
    if ($ok) { $ok = mysqli_stmt_execute($req_maj); }
    if (! $ok) { $message = mysqli_stmt_error($req_maj);}
    mysqli_stmt_close($req_maj);
  } else {
    $message = mysqli_error($db);
  }
```

```
// Enregistrement des rubriques du livre.
// Annule (DELETE) et remplace (INSERT) l'existant.
if ($ok) {
  // Suppression des rubriques actuelles.
  $sql = 'DELETE FROM rubrique_livre WHERE id_livre = ?';
  if ($ok = ($req_maj = mysqli_prepare($db,$sql))) {
    $ok = mysqli_stmt_bind_param($req_maj,'i',$id_livre);
    if ($ok) { $ok = mysqli_stmt_execute($req_maj); }
    if (! $ok) { $message = mysqli_stmt_error($req_maj);}
    mysqli_stmt_close($req_maj);
  } else {
    $message = mysqli_error($db);
  }
  // Insertions des nouvelles rubriques (s'il y en a).
  if ($ok AND ($rubriques = $livre['rubriques'])) {
    $sql = 'INSERT INTO rubrique_livre(id_livre,id_rubrique) '.
           'VALUES(?,?)';
    if ($ok = ($req_maj = mysqli_prepare($db,$sql))) {
      $ok = mysqli_stmt_bind_param
               ($req_maj,'ii',$_POST['id'],$id_rubrique);
      if ($ok) {
        foreach ($rubriques as $id_rubrique) {
          $ok = mysqli_stmt_execute($req_maj);
          if (! $ok) {
            $message = mysqli_stmt_error($req_maj);
            break;
          }
        }
      }
      mysqli_stmt_close($req_maj);
    } else {
      $message = mysqli_error($db);
    }
  }
}
// Enregistrement de l'image de couverture
if ($ok) {
  // Si un fichier a été téléchargé avec succès, lire
  // son contenu.
  switch ($_FILES['couverture']['error']) {
    case UPLOAD_ERR_NO_FILE :
      break;
    case UPLOAD_ERR_OK :
      $couverture =
        file_get_contents($_FILES['couverture']['tmp_name']);
      if (! $couverture) {
        $message = 'problème avec l\'image de couverture';
      }
      break;
    default :
      $message = 'image de couverture non transférée';
      break;
  }
  if ($couverture) { // il a bien une image ...
    $sql = 'UPDATE livre SET couverture = ? WHERE id = ?';
    if ($ok = $req_maj = mysqli_prepare($db,$sql)) {
      $ok = mysqli_stmt_bind_param
               ($req_maj,'bi',$couverture,$_POST['id']);
      if ($ok) {
        $ok = mysqli_stmt_send_long_data
                 ($req_maj,0,$couverture);
      }
      if ($ok) { $ok = mysqli_stmt_execute($req_maj); };
      if (! $ok) { $message = mysqli_stmt_error($req_maj);}
      mysqli_stmt_close($req_maj);
    } else {
      $message = mysqli_error($db);
    }
  }
```

```
  }
  // Compléter le message d’erreur si nécessaire.
  if ($message) {
    $message =
      "Erreur lors de l’enregistrement du livre ($message).";
  }
}
// S’il y a un livre à charger ...
if ($charger_livre) {
  // Charger les informations sur le livre proprement dit,
  // dans le tableau associatif $livre.
  $sql = "SELECT *  FROM livre WHERE id = $id_livre";
  if ($ok = ($req_liv = mysqli_query($db,$sql))) {
    $livre = mysqli_fetch_assoc($req_liv);
    if (! $livre) {
      $message = ’Aucun livre trouvé.’;
    }
  }
  // Si tout est OK à ce stade, sélectionner la liste
  // des collections (pour la liste déroulante).
  // Le fetch sera fait plus tard.
  if ($ok AND $livre) {
    $sql = ’SELECT id,nom FROM collection’;
    $ok = ($req_col = mysqli_query($db,$sql));
  }
  // Si tout est OK à ce stade, sélectionner la liste
  // des rubriques (pour la liste de sélection).
  // Le fetch sera fait plus tard.
  if ($ok AND $livre) {
    // La requête utilisée permet d’avoir une colonne
    // ’selection’ qui est à 1 lorsque la rubrique est
    // sélectionnée pour le livre.
    $sql =
      "SELECT rub.id,rub.titre,rul.selection
      FROM
        ( /* liste des sous-rubriques, sous la forme
            ’<rubrique> - <sous-rubrique>’ */
        SELECT sru.id,CONCAT(rub.titre,’ - ’,sru.titre) titre
        FROM rubrique sru JOIN rubrique rub
             ON (sru.id_parent = rub.id)
        WHERE sru.id_parent IS NOT NULL
        ) rub
        LEFT JOIN
        ( /* liste des rubriques sélectionnées pour le livre */
        SELECT id_rubrique,1 selection
        FROM rubrique_livre
        WHERE id_livre = $id_livre
        ) rul
        ON (rub.id = rul.id_rubrique)";
    $ok = ($req_rub = mysqli_query($db,$sql));
  }
  // En cas d’erreur, initialisation d’un message et effacement
  // du tableau $livre.
  if (! $ok) {
    $message = ’Erreur lors du chargement du livre.’;
    unset($livre);
  }
}
// Affichage de la page ...
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Saisir un livre</title></head>
  <body>
    <!-- Formulaire de saisie de l’identifiant. -->
    <form action="saisie-livre.php" method="post">
    <div>
    Identifiant :
```

```
<input type="text" name="id" size="6"
   value="<?php echo vers_formulaire($id_livre); ?>" />
<input type="submit" name="charger" value="Charger" />
</div>
</form>
<!-- Formulaire de saisie du livre (affiché uniquement si
     un livre a été chargé avec succès) -->
<?php if ($livre): ?>
<form action="saisie-livre.php" method="post"
 enctype="multipart/form-data">
<div>
  <br />ISBN :
  <input type="text" name="isbn" size="20" maxlength="20"
   value="<?php echo vers_formulaire($livre['isbn']); ?>" />
  <br />Titre :
  <input type="text" name="titre" size="75" maxlength="75"
   value="<?php echo vers_formulaire($livre['titre']); ?>" />
  <br />Sous-titre :
  <input type="text" name="sous_titre" size="75" maxlength="75"
   value=
    "<?php echo vers_formulaire($livre['sous_titre']); ?>" />
  <br />Nombre pages :
  <input type="text" name="nombre_pages" size="4" maxlength="4"
   value=
    "<?php echo vers_formulaire($livre['nombre_pages']); ?>" />
  Année de parution :
  <input type="text" name="annee_parution"
   size="4" maxlength="4"
   value=
   "<?php echo vers_formulaire($livre['annee_parution']); ?>"
  />
  <br />Niveau :
  <?php
  // Génération dynamique des boutons radios utilisés pour
  // le niveau.
  $niveaux = array('Débutant','Initié','Confirmé','Expert');
  foreach ($niveaux as $niveau) {
    // Niveau du livre = niveau courant => bouton coché.
    if ($livre['niveau'] == $niveau) {
      $sélection = 'checked="checked"';
    } else {
      $sélection = '';
    }
    printf
      (
      '<input type="radio" name="niveau" %s value="%s" />%s',
      $sélection,$niveau,$niveau
      );
  }
  ?>
  <br />Collection :
  <select name="collection">
  <?php
  // Génération dynamique de la liste utilisée pour
  // la collection.
  while ($collection = mysqli_fetch_assoc($req_col)) {
    // Collection du livre = collection courante
    // => ligne sélectionnée.
    if ($livre['id_collection'] == $collection['id']) {
      $sélection = 'selected="selected"';
    } else {
      $sélection = '';
    }
    printf
      ('<option %s value="%s">%s</option>',
       $sélection,$collection['id'],$collection['nom']);
  }
  ?>
  </select>
```

```
      <br />Rubriques :<br />
      <select name="rubriques[]" multiple="multiple" size="8">
      <?php
      // Génération dynamique de la liste utilisée pour
      // les rubriques.
      while ($rubrique = mysqli_fetch_assoc($req_rub)) {
        // Colonne 'selection' = 1 = rubrique du livre
        // => ligne sélectionnée.
        if ($rubrique['selection'] == 1) {
          $sélection = 'selected="selected"';
        } else {
          $sélection = '';
        }
        printf
          ('<option %s value="%s">%s</option>',
           $sélection,$rubrique['id'],$rubrique['titre']);
      }
      ?>
      </select>
      <br />Description :<br />
      <textarea name="description" rows="6" cols="65"><?php
      echo vers_formulaire($livre['description']); ?></textarea>
      <br />
      <!-- L'image de couverture est affichée par appel à un
           autre script PHP. -->
      <img alt="Couverture :"
       src="image-livre.php?id=<?php echo $id_livre; ?>" />
      <input type="file" name="couverture" value="" size="65" />
      <input type="hidden" name="id"
       value="<?php echo $id_livre; ?>" />
      <br />
      <input type="submit" name="ok" value="Enregistrer" />
    </div>
    </form>
    <?php endif; ?>
    <div><?php echo vers_page($message); ?></div>
  </body>
</html>
```

Le formulaire actuel permet uniquement de modifier un livre existant. Il peut être facilement modifié pour permettre la saisie d'un nouveau livre.

# Description du problème

Le protocole HTTP (*HyperText Transfer Protocol*) est un protocole "sans état" : rien ne permet d'identifier que c'est le même utilisateur qui était précédemment sur la page A et qui maintenant accède à la page B.

En ce qui concerne PHP, nous avons vu qu'une variable a une portée égale au script dans lequel elle est définie et n'existe que le temps de l'exécution d'un script.

Or, un site interactif qui ne se contente pas d'afficher des pages les unes derrière les autres, a souvent besoin, du point de vue de la logique applicative, d'identifier un utilisateur d'une page à l'autre et de conserver des informations relatives à cet utilisateur d'une page à l'autre (typiquement, un panier électronique constitué par l'utilisateur sur une page doit toujours être défini sur la page permettant le paiement).

Le terme "session" désigne la période de temps correspondant à la navigation continue d'un utilisateur sur un site. "Gérer les sessions" consiste donc à être en mesure d'identifier l'instant où un nouvel utilisateur accède à une page du site et de conserver des informations relatives à cet utilisateur jusqu'à ce qu'il quitte le site. L'utilisateur n'est pas forcément un utilisateur authentifié par un nom et un mot de passe mais peut très bien être un "anonyme", non référencé par le site, qui effectue un achat. Beaucoup de sites interactifs proposent des fonctionnalités d'identification (membre, abonné ...) car cela permet de conserver des informations sur l'utilisateur d'une visite à l'autre (préférences par exemple). Cette possibilité sera aussi étudiée dans ce chapitre mais du point de vue de la notion de session, la visite de l'utilisateur le vendredi correspondra à une session différente de sa visite du lundi, même si certaines informations saisies le lundi sont susceptibles d'être restituées le vendredi.

Ce chapitre a pour objectif de présenter les différentes techniques qui vont permettre, d'une part d'identifier un utilisateur et d'autre part de "suivre" cet utilisateur, et les données qui lui sont associées, d'une page à l'autre.

En préambule, nous allons voir comment authentifier un utilisateur.

Enfin, nous allons terminer ce chapitre en évoquant les techniques qui permettent de conserver des informations d'une visite à l'autre.

Dans le chapitre Gérer les formulaires et les liens avec PHP, nous avons vu comment passer des informations d'une page à l'autre à l'aide d'un formulaire (avec la possibilité d'utiliser des zones cachées si besoin) ou d'une URL. Ces techniques peuvent être utilisées pour gérer les sessions, mais c'est un peu "artisanal" ; il est préférable d'utiliser les fonctionnalités natives de PHP pour gérer les sessions.

# Authentification

## 1. Vue d'ensemble

Certains sites ont besoin d'authentifier les utilisateurs qui accèdent au site afin de vérifier que ces derniers sont bien inscrits.

Cette authentification comprend généralement deux étapes :

- saisie par l'utilisateur d'informations d'identification, typiquement un nom et un mot de passe ;
- vérification que l'identification saisie correspond bien à un utilisateur inscrit.

## 2. Saisie de l'identification

L'identification peut être saisie de deux manières :

- par l'intermédiaire d'un formulaire prévu à cet effet ;
- par les fonctions d'authentification HTTP.

### a. Identification par formulaire

Il est très simple de créer un petit formulaire permettant à l'utilisateur de saisir un nom et un mot de passe.

*Exemple de script PHP (`login.php`) qui affiche ce formulaire (fonction de vérification pour l'instant non définie)*

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include('fonctions.inc');
// Fonction qui vérifie que l'identification saisie
// est correcte.
function utilisateur_existe($identifiant,$mot_de_passe) {
   // Aléatoire, en attendant mieux ...
   return (bool) rand(0,1);
}
// Initialisation des variables.
$identifiant = '';
$mot_de_passe = '';
$message = '';
// Traitement du formulaire.
if (isset($_POST['connexion'])) {
   // Récupérer les informations saisies.
   $identifiant = valeur_saisie($_POST['identifiant']);
   $mot_de_passe = valeur_saisie($_POST['mot_de_passe']);
   // Vérifier que l'utilisateur existe.
   if (utilisateur_existe($identifiant,$mot_de_passe)) {
      // L'utilisateur existe ...
      // Partir sur une autre page et interrompre
      // le script.
      header('location: accueil.php');
      exit;
   } else {
      // L'utilisateur n'existe pas ...
      // Afficher un message et proposer de
      // nouveau l'identification
      $message  = 'Identification incorrecte. ';
      $message .= 'Essayez de nouveau.';
      // Laisser le formulaire s'afficher de nouveau ...
   }
}
```

```
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>MonSite.com</title></head>
  <body>
    <form action="login.php" method="post">
    <table border="0">
    <tr>
      <td align="right">Identifiant :</td>
      <td><input type="text" Name="identifiant" value=
           "<?php echo vers_formulaire($identifiant); ?>" /></td>
    </tr>
    <tr>
      <td align="right">Mot de passe :</td>
      <td><input type="password" Name="mot_de_passe" value=
           "<?php echo vers_formulaire($mot_de_passe); ?>" /></td>
    </tr>
    <tr>
      <td></td>
      <td align="right"><input type="submit" name="connexion"
                          value="Connexion" /></td>
    </tr>
    </table>
    <?php echo $message; ?>
    </form>
  </body>
</html>
```

*Résultat*

- Affichage initial :

Identifiant :
Mot de passe :
Connexion

- Saisie :

Identifiant : heurtel
Mot de passe : *******
Connexion

- Résultat si l'identification est erronée :

Identifiant : heurtel
Mot de passe : *******
Connexion
Identification incorrecte. Essayez de nouveau.

L'utilisation d'une zone de type password permet de masquer la saisie du mot de passe.

Si la saisie est incorrecte, la page est proposée de nouveau. Dans le cas contraire, une page d'accueil est affichée.

### b. Identification par authentification HTTP

À l'aide de la fonction header (cf. Utiliser les fonctions PHP - Manipuler les en-têtes HTTP), il est possible de demander

au navigateur d'afficher une fenêtre de dialogue invitant l'utilisateur à saisir un nom et un mot de passe. Le message d'en-tête à envoyer est :

```
WWW-Authenticate: Basic realm="xxxxx"
```

*Avec*

xxxxx = nom affiché (nom d'organisation par exemple)

Si l'utilisateur clique sur le bouton **OK**, le script est appelé de nouveau avec les valeurs saisies disponibles dans les variables PHP $PHP_AUTH_USER et $PHP_AUTH_PW.

Par sécurité, il est préférable de récupérer ces informations par l'intermédiaire du tableau associatif $_SERVER, avec les clés PHP_AUTH_USER et PHP_AUTH_PW.

*Exemple de script login.php utilisant cette technique*

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include('fonctions.inc');
// Fonction qui vérifie que l'identification saisie
// est correcte.
function utilisateur_existe($identifiant,$mot_de_passe) {
  // Aléatoire, en attendant mieux ...
  return (bool) rand(0,1);
}
// Fonction qui affiche l'authentification HTTP.
function authentification($message) {
  header("WWW-Authenticate: Basic realm=\"$message\"");
  // Si l'utilisateur clique sur le bouton annuler,
  // Les lignes suivantes s'exécutent (sinon, le script est
  // de nouveau appelé mais avec $PHP_AUTH_USER renseigné
  // et le script ne passera plus par ici).
  // Afficher un message et proposer à l'utilisateur
  // d'essayer de nouveau
  echo 'Vous devez saisir un nom et un mot de passe ',
       'pour accéder au site.<br />';
  echo '<a href="login.php">Essayer de nouveau</a>';
  exit;
}
if (! isset($_SERVER["PHP_AUTH_USER"])) {
  // Pas de variable $PHP_AUTH_USER = premier appel du script.
  // Demande d'identifiation.
  authentification("MonSite.com");
} else {
  // Variable $PHP_AUTH_USER existe = appel après saisie.
  // Récupérer les information saisies.
  $identifiant = valeur_saisie($_SERVER["PHP_AUTH_USER"]);
  $mot_de_passe = valeur_saisie($_SERVER["PHP_AUTH_PW"]);
  // Vérifier que l'utilisateur existe.
  if (utilisateur_existe($identifiant,$mot_de_passe)) {
    // L'utilisateur existe ...
    // Partir sur une autre page et interrompre le script
    header('location: accueil.php');
    exit;
  } else {
    // L'utilisateur n'existe pas ...
    // Essayer de nouveau.
    authentification('MonSite.com : identification incorrecte');
  }
}
?>
```

*Résultat*

- Affichage initial (Firefox) :

**Authentification requise**

Entrez le nom et le mot de passe pour MonSite.com à http://xampp

Utilisateur :

Mot de passe :

☐ Utiliser le gestionnaire de mots de passe pour se souvenir de ce mot de passe.

OK Annuler

- Clic sur le bouton **Annuler** :

Vous devez saisir un nom et un mot de passe pour accéder au site.
Essayer de nouveau

- Nouvel affichage (par le lien) et saisie :

**Authentification requise**

Entrez le nom et le mot de passe pour MonSite.com à http://xampp

Utilisateur :

heurtel

Mot de passe :

********

☐ Utiliser le gestionnaire de mots de passe pour se souvenir de ce mot de passe.

OK Annuler

- Nouvel affichage si l'identification est incorrecte :

**Authentification requise**

Entrez le nom et le mot de passe pour MonSite.com : identification incorrecte à http://xampp

Utilisateur :

Mot de passe :

☐ Utiliser le gestionnaire de mots de passe pour se souvenir de ce mot de passe.

OK Annuler

Si l'identification est correcte, une page d'accueil est affichée.

---

Pour l'instant, sur les deux exemples, l'accès à la page d'accueil n'est pas protégé : un utilisateur qui demande cette page y accède sans problème.

---

## 3. Vérifier l'identification saisie

Quelle que soit la méthode utilisée au point précédent, il convient ensuite de vérifier que les informations saisies correspondent bien à un utilisateur "connu".

Typiquement, ce contrôle sera réalisé à l'aide d'une base de données qui contient notamment la liste des utilisateurs.

Pour la suite, nous supposerons qu'il existe dans une base MySQL, une table `utilisateurs` comportant deux colonnes, `identifiant` et `mot_de_passe`.

*Exemple*

```
<?php
// Fonction qui vérifie que l'identification saisie
// est correcte.
function utilisateur_existe($identifiant,$mot_de_passe) {
  // Connexion et sélection de la base de données
  $connexion = mysqli_connect('localhost','root');
  mysqli_select_db($connexion,'eni');
  // Définition et exécution d'une requête préparée
  $sql  = 'SELECT 1 FROM utilisateurs ';
  $sql .= 'WHERE identifiant = ? AND mot_de_passe = ?';
  $requête = mysqli_stmt_init($connexion);
  $ok = mysqli_stmt_prepare($requête,$sql);
  $ok = mysqli_stmt_bind_param
          ($requête,'ss',$identifiant,$mot_de_passe);
  $ok = mysqli_stmt_execute($requête);
  mysqli_stmt_bind_result($requête,$existe);
  $ok = mysqli_stmt_fetch($requête);
  mysqli_stmt_free_result($requête);
  // L'identification est bonne si la requête a retourné
  // une ligne (l'utilisateur existe et le mot de passe
  // est bon).
  // Si c'est le cas $existe contient 1, sinon elle est
  // vide. Il suffit de la retourner en tant que booléen.
  return (bool) $existe;
}?>
```

D'autre méthodes d'authentification, qui ne s'appuient pas sur une base de données sont envisageables (simple fichier par exemple).

# Utiliser des cookies

## 1. Principe

Un cookie est un petit fichier déposé, par un site, sur le poste de l'internaute et qui peut contenir des informations. Les cookies sont automatiquement renvoyés au serveur Web, par le navigateur, lorsque l'internaute navigue dans les pages du site en question.

PHP permet de récupérer très facilement, dans des variables, les données stockées dans le cookie.

La fonction `setcookie` permet de déposer un cookie sur le poste de l'internaute.

<u>*Syntaxe*</u>

```
booléen setcookie(chaîne nom [, chaîne valeur [, entier
expiration [, chaîne chemin [, chaîne domaine [, booléen
securisé[, booléen http_uniquement]]]]]])
```

`nom`

Nom du cookie.

`valeur`

Valeur stockée dans le cookie.

`expiration`

Date d'expiration du cookie (*timestamp* Unix)

`chemin`

Chemin du répertoire sur le serveur dans lequel le cookie est disponible. Mettre `/` pour rendre le cookie disponible sur le domaine entier ou `/rep/` pour rendre le cookie disponible dans le répertoire `/rep/` du domaine et tous ses sous-répertoires. Par défaut, égal au répertoire à partir duquel le cookie a été déposé.

`domaine`

Domaine auquel le cookie est renvoyé. `.monSite.com` (avec un point au début) permet par exemple de rendre le cookie disponible pour tous les sous-domaines de `monSite.com`.

`sécurisé`

Mettre `TRUE` pour indiquer que le cookie ne doit être transmis que sur une connexion sécurisée (`FALSE` par défaut).

`http_uniquement`

Mettre `TRUE` pour indiquer que le cookie ne doit être transmis que pour le protocole HTTP (`FALSE` par défaut). Ajouté en version 5.2.0.

Si la fonction n'est appelée qu'avec le paramètre `nom`, le cookie portant ce nom est supprimé du poste de l'internaute. Si les paramètres `domaine` et `chemin` avaient été spécifiés lors du dépôt du cookie, il faut les spécifier à l'identique pour supprimer le cookie (mettre une date d'expiration dans le passé).

Si le paramètre `valeur` est spécifié, un cookie portant le nom `nom` et contenant la valeur `valeur` est envoyé sur le poste de l'utilisateur ; s'il existe déjà un cookie portant ce nom, ce dernier est mis à jour avec la nouvelle valeur.

Le paramètre `expiration` permet de déterminer la date d'expiration du cookie (et donc la date de sa suppression du poste de l'utilisateur) ; si ce paramètre est non spécifié (ou égal à 0) le cookie expire à la fin de la session, c'est-à-dire lorsque l'utilisateur quitte le site.

Les cookies sont envoyés dans l'en-tête de la page. À l'instar de la fonction `header`, la fonction `setcookie` doit donc être appelée avant toute instruction (PHP ou HTML) qui a pour effet de commencer à construire la page HTML. En cas de problème, un message du type suivant est affiché :

```
Warning: Cannot add header information - headers already sent
by (output started at /app/scripts/test.php:1) in /app/scripts/test.php on line 7
```

La fonction setcookie retourne TRUE si l'instruction a pu être exécutée (pas de données déjà transmises) et FALSE dans le cas contraire. Par contre, le code de retour de la fonction ne donne aucune information sur le fait que le cookie a réellement pu être déposé sur le poste de l'utilisateur : si ce dernier refuse les cookies, la fonction setcookie retourne quand même TRUE bien que le cookie n'ait pas été déposé.

Les cookies sont gérés par site ; deux cookies de sites différents peuvent porter le même nom. Le cookie est déposé sur le poste de l'internaute par la fonction setcookie puis renvoyé ultérieurement lors de la visite de n'importe quelle page du site.

*Exemple*

```
<?php
// Dépôt d'un cookie nommé "nom" contenant
// la valeur "Olivier" et expirant à la fin de
// la session.
$ok = setcookie('nom','Olivier');
// Idem mais expirant à date du jour (time() en secondes)
// plus 30 fois 24 fois 3600 secondes (soit 30 jours).
$ok = setcookie('nom','Olivier',time()+(30*24*3600));
// Suppression du cookie nommé 'nom'.
$ok = setcookie('nom');
?>
```

Lorsque le cookie est renvoyé au serveur Web, par le navigateur, lors de la demande d'une page PHP, la valeur du cookie est accessible dans une variable PHP selon un mécanisme similaire à celui mis en œuvre pour les formulaires et les URL.

La valeur de chaque cookie envoyé par le navigateur est automatiquement enregistrée dans le tableau associatif $_COOKIE : la clé du tableau est égale au nom du cookie.

Les variables de cookie sont aussi disponibles dans le tableau associatif $_REQUEST ou peuvent être importées dans le script par appel à la fonction import_request_variables (cf. chapitre Gérer les formulaires et les liens avec PHP - Vue d'ensemble).

*Exemple*

- Script page1.php qui dépose deux cookies :

```
<?php
// Premier cookie expirant à la fin de la session.
$ok1 = setcookie('prénom','Olivier');
// Deuxième cookie expirant dans 30 jours.
$ok2 = setcookie('nom','HEURTEL',time()+(30*24*3600));
// Résultat.
if ($ok1 and $ok2) {
   $message = 'Cookies déposés (du moins, a priori)';
} else {
   $message = 'L\'un des cookies n\'a pas pu être déposé';
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <?php echo $message; ?><br />
    <!-- lien vers la page 2 -->
    <a href="page2.php">Page 2</a>
    </div>
  </body>
</html>
```

- Script page2.php qui affiche la valeur des deux cookies :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 2</title></head>
  <body>
    <div>
<?php
if ( isset($_COOKIE["prénom"]) ) {
      echo "\$_COOKIE[\"prénom\"] = {$_COOKIE['prénom']}<BR>";
} else {
      echo "\$_COOKIE[\"prénom\"] = <BR>";
}
if ( isset($_COOKIE["nom"]) ) {
      echo "\$_COOKIE[\"nom\"] = {$_COOKIE['nom']}<BR>";
} else {
      echo "\$_COOKIE[\"nom\"] = <BR>";
}
?>
    </div>
  </body>
</html>
```

*Résultat*

- Affichage de la page 1




- Résultat du clic sur le lien




- Résultat d'un retour, avant trente jours, sur la page 2 du même site :




Le cookie de durée de vie égale à la session n'existe plus et l'information est perdue ; l'information stockée dans l'autre cookie reste disponible (dans la limite de sa durée de vie).

Il est possible de stocker n'importe quelle chaîne dans le cookie sans avoir à se soucier d'un éventuel encodage/décodage : l'encodage et le décodage sont effectués automatiquement.

Le cookie est déposé sur le poste de l'internaute par la fonction `setcookie` puis renvoyé ultérieurement lors de la visite de n'importe quelle page du site ; le cookie n'est pas disponible immédiatement dans la page qui le dépose.

*Exemple*

```
<?php
// valeur du cookie avant.
$avant = (isset($_COOKIE ['heure']))?$_COOKIE ['heure']:'';
// Dépôt du cookie expirant à la fin de la session.
$ok = setcookie('heure',date('H:i:s'));
// Valeur du cookie après.
$après = (isset($_COOKIE ['heure']))?$_COOKIE ['heure']:'';
// heure actuelle
$actuel = date('H:i:s');
// Affichage
echo "Actuel : $actuel<br />";
echo "Avant : $avant<br />";
echo "Après : $après<br />";
?>
```

*Résultat du premier appel*

```
Actuel : 08:28:55
Avant :
Après :
```

Lors du premier appel, le cookie n'existe pas avant (c'est normal) et n'existe toujours pas après, car il a simplement été envoyé mais n'est pas encore "revenu" (manière imagée de présenter les choses).

*Résultat du deuxième appel (dans la même session)*

```
Actuel : 08:29:19
Avant : 08:28:55
Après : 08:28:55
```

Lors du deuxième appel, la valeur du cookie est disponible dès le début du script (le cookie est revenu avec la requête pour la page) et a bien une valeur qui correspond à l'instant où il a été déposé ; par contre sa valeur après ne reflète pas immédiatement la réalité (même principe que pour le dépôt initial : le cookie n'est pas encore "revenu").

Une des conséquences de ce mode de fonctionnement est qu'il n'est pas possible, tout de suite après avoir déposé le cookie, de tester si le cookie a été accepté ou non par le poste.

Pour savoir si un poste accepte les cookies, il faut déposer un cookie et recharger une page dans laquelle la présence ou non du cookie permettra de déterminer si le poste accepte les cookies.

*Exemple de script `tester_cookie.php` qui permet de faire ce test*

```
<?php
// Tester si c'est le deuxième appel de la page.
if (! isset($_GET['retour'])) {
  // Non ...
  // Déposer le cookie.
```

```
  setcookie('test','test');
  // Et recharger la page avec une information dans
  // l'URL indiquant que c'est le deuxième passage.
  header('Location: tester_cookie.php?retour=1');
} else {
  // Oui ...
  // Tester si le cookie est "revenu".
  if (isset($_COOKIE['test'])) { // oui ...
    echo 'Cookie accepté';
  } else { // non ...
    echo 'Cookie refusé';
  }
}
?>
```

Il est possible de récupérer un tableau comme valeur de cookie sous réserve d'utiliser une notation de type tableau lors du dépôt du cookie.

*Exemple (basé sur l'exemple précédent)*

```
<?php
// Inclusion du fichier qui contient les fonctions générales.
include('fonctions.inc');
// Tester si c'est le deuxième appel de la page.
if (! isset($_GET['retour'])) {
  // Non ...
  // Déposer le cookie.
  setcookie('test[0]','zéro');
  setcookie('test[1]','un');
  // Et recharger la page avec une information dans
  // l'URL indiquant que c'est le deuxième passage.
  header('Location: tester_cookie.php?retour=1');
} else {
  // Oui ...
  // Tester si le cookie est "revenu".
  if (isset($_COOKIE['test'])) { // oui ...
    echo 'Cookie accepté<br />';
    afficher_tableau($_COOKIE['test']);
  } else { // non ...
    echo 'Cookie refusé';
  }
}
?>
```

*Résultat*

```
Cookie accepté
0 = zéro
1 = un
```

Dans la pratique, il y a en fait plusieurs cookies déposés sur le poste de l'utilisateur, mais les valeurs sont bien récupérées dans le script PHP sous la forme d'un tableau. Pour déposer un seul cookie contenant plusieurs valeurs, vous pouvez procéder par concaténation, ou utiliser une fonction comme `implode` (et `explode` au retour du cookie).

## 2. "magic quotes" : le retour

Comme pour les données `GET` et `POST`, la valeur récupérée dans un cookie peut subir l'encodage "magic quotes" si la directive de configuration `magic_quotes_gpc` (*Get/Post/Cookie*) est à `on` (cf. chapitre Utiliser les fonctions PHP - Gérer les "guillemets magiques" ("magic quotes")).

En conséquence, lors de la récupération d'une information transmise par URL, il convient éventuellement d'appeler la fonction `stripslashes` ou nos fonctions génériques `valeur_saisie` ou `supprimer_encodage_MQ_GPC` pour supprimer l'encodage "magic quotes".

## 3. Application à la gestion des sessions

Les cookies peuvent être utilisés pour gérer les sessions, avec l'avantage d'être parfaitement indépendants à la fois de la navigation par balise `<a href=...>` et de la gestion des formulaires.

Ils présentent par contre un gros inconvénient : ils peuvent être refusés par les internautes. Par ailleurs, le nombre de cookies est limité à 20 par serveur et la taille d'un cookie est elle aussi limitée.

Une gestion des sessions, dans laquelle un identifiant de session est transmis d'une page à l'autre (les autres informations étant stockées sur le serveur) peut être mise en place relativement facilement en utilisant les cookies, si l'utilisateur les accepte, et en utilisant l'URL dans le cas contraire. C'est exactement ce que propose la gestion des sessions de PHP que nous allons voir maintenant : ne nous amusons donc pas à redévelopper ce qui existe déjà !

Nous allons voir également dans ce chapitre (cf. Conserver des informations d'une visite à une autre), que le cookie est un bon outil (sous réserve de l'accord de l'utilisateur) pour stocker une information d'une session à l'autre (en utilisant donc un cookie ayant une durée de vie spécifiée à la création).

# Utiliser la gestion des sessions de PHP

## 1. Principes

Depuis la version 4, PHP propose un ensemble de fonctions qui facilitent la gestion des sessions. Les principes sont les suivants :

- Un identifiant unique est automatiquement attribué à chaque session.
- Cet identifiant unique est transmis d'une page à l'autre, soit par cookie (si le poste accepte les cookies), soit par l'URL dans le cas contraire ; en tout état de cause, c'est PHP qui choisit automatiquement la bonne méthode et assure ce transfert (à quelques réserves près liées à la configuration).
- Les données dont vous souhaitez conserver la valeur d'une page à l'autre pendant la durée de la session sont indiquées à PHP qui se charge automatiquement de restituer leur valeur au début du script et de les sauvegarder à la fin du script.

En bref, PHP se charge de toute la gestion.

## 2. Mise en œuvre

Les principales fonctions du module de gestion des sessions sont les suivantes :

| Nom | Rôle |
|---|---|
| `session_start` | Ouvre une nouvelle session ou réactive la session courante. |
| `session_id` | Retourne (ou éventuellement modifie) l'identifiant de la session. |
| `session_name` | Retourne (ou éventuellement modifie) le nom de la variable utilisée pour stocker l'identifiant de la session. |
| `session_destroy` | Supprime la session. |

En complément, le tableau `$_SESSION` permet de manipuler très facilement les variables de session.

Plusieurs fonctions (notamment `session_register` et `session_unregister`) sont conservées par PHP pour des raisons de compatibilité ascendante et ne fonctionnent que lorsque la directive de configuration `register_globals` est à `on`. L'utilisation de `$_SESSION` est recommandée et rend ces fonctions inutiles ; elles ne sont donc pas présentées dans cet ouvrage.

N'oubliez pas que la directive de configuration `register_globals` est à `off` par défaut depuis la version 4.2 et qu'elle passera définitivement à `off` dans une prochaine version.

### session_start

*Syntaxe*

```
booléen session_start()
```

La fonction `session_start` interroge l'environnement pour détecter si une session a déjà été ouverte pour l'utilisateur actuel. Si oui, les données enregistrées dans la session sont restituées. Autrement, une nouvelle session est ouverte avec attribution d'un identifiant.

La fonction `session_start` retourne toujours `TRUE`.

Tout script concerné par la gestion des sessions doit appeler `session_start` pour pouvoir avoir accès aux variables de

session.

Si la session n'est pas encore ouverte, la fonction session_start va chercher à déposer un cookie, contenant l'identifiant de session, sur le poste de l'utilisateur : il est donc primordial, comme pour les fonctions header et setcookie que le début de la page n'ait pas encore été envoyé au navigateur. En cas de problème, un message du type suivant est affiché :

```
Warning: session_start() [function.session-start]: Cannot
send session cookie - headers already sent by (output started
at /app/scripts/test.php:1) in /app/scripts/test.php on line 4
```

*Exemple*

```
<?php
// Ouvrir/réactiver la session.
session_start();
?>
```

## session_id

*Syntaxe*

```
chaîne session_id([chaîne nouvelle_valeur])
```

nouvelle_valeur

Nouvelle valeur attribuée à l'identifiant de session.

Appelée sans paramètre, la fonction session_id retourne la valeur de l'identifiant de session. Cette valeur sera toujours vide si session_start n'a pas été appelé dans le script.

Appelée avec un paramètre, la fonction session_id modifie la valeur attribuée à l'identifiant de session. En pratique, dans la majorité des cas, cela ne présente pas un grand intérêt.

*Exemple*

```
<?php
// Récupérer la valeur de session_id avant.
$avant = session_id();
// Ouvrir/réactiver la session.
session_start();
// Récupérer la valeur de session_id après.
$après = session_id();
// Affichage.
$actuel = date("H:i:s");
echo "Heure : $actuel<br />";
echo "Avant : $avant<br />";
echo "Après : $après<br />";
?>
```

*Résultat*

- Premier appel

```
Heure : 20:15:40
Avant :
Après : 3abfb1ff70a7dc7ad13d1d47d890880e
```

- Deuxième appel sans quitter le site (même session)

```
Heure : 20:15:56
Avant :
Après : 3abfb1ff70a7dc7ad13d1d47d890880e
```

## session_name

*Syntaxe*

```
chaîne session_name([chaîne nouvelle_valeur])
```

`nouvelle_valeur`

Nouvelle valeur attribuée au nom de la variable qui stocke l'identifiant de session.

Appelée sans paramètre, la fonction `session_name` retourne le nom de la variable dans laquelle l'identifiant de session est stocké. La fonction `session_name` retourne un résultat même si la fonction `session_start` n'a pas été appelée dans le script.

Appelée avec un paramètre, la fonction `session_name` modifie le nom de la variable. En pratique, dans la majorité des cas, cela ne présente pas un grand intérêt (il est de toute façon remis à sa valeur par défaut à la fin du script).

*Exemple*

```
<?php
// Récupérer la valeur de session_name avant.
$avant = session_name();
// Ouvrir/réactiver la session.
session_start();
// Récupérer la valeur de session_name après.
$après = session_name();
// Affichage.
echo "Avant : $avant<br />";
echo "Après : $après<br />";
// Affichage de la valeur.
// L'identifiant de session est transmis soit par cookie,
// soit par l'URL, et donc disponible en tant que variable
// dans le tableau $_REQUEST.
echo "$après = {$_REQUEST[$après]}<br />";
// Comparaison avec la valeur retournée par session_id().
echo 'session_id() = ',session_id();
?>
```

*Résultat*

```
Avant : PHPSESSID
Après : PHPSESSID
PHPSESSID = 3abfb1ff70a7dc7ad13d1d47d890880e
session_id() = 3abfb1ff70a7dc7ad13d1d47d890880e
```

Le nom de la variable est défini par la directive de configuration `session.name`. Vous pouvez modifier cette directive pour utiliser un autre nom.

### Manipuler les variables enregistrées dans la session

Après appel à la fonction `session_start`, les données de session peuvent être manipulées directement dans le tableau associatif `$_SESSION`. Toutes les entrées stockées dans le tableau `$_SESSION` sont automatiquement enregistrées en tant que données de session.

Pour enregistrer une nouvelle donnée dans la session, il suffit de stocker cette donnée dans le tableau `$_SESSION`, avec la clé de votre choix.

Pour lire ou modifier une donnée de session préalablement enregistrée; il suffit d'accéder au tableau `$_SESSION` en utilisant la bonne clé.

---

N'oubliez pas d'appeler la fonction `session_start` pour pouvoir manipuler les données de session à l'aide du tableau `$_SESSION`.

---

- Script `page1.php` qui ouvre une session et enregistre des données dans la session.

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Enregistrer deux informations dans la session.
$_SESSION['prénom'] = 'Olivier';
$_SESSION['informations'] =  // c'est un tableau ...
```

```
      array('prénom'=>'Olivier','nom'=>'HEURTEL');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div><a href="page2.php">Page 2</a></div>
  </body>
</html>
```

- Script page2.php qui affiche la valeur des variables de session.

```
<?php
// Appel à session_start.
session_start();
// Affichage.
echo '$_SESSION[\'prénom\'] = ',
     isset($_SESSION['prénom'])?$_SESSION['prénom']:'',
     '<br />';
echo '$_SESSION[\'informations\'][\'nom\'] = ',
     isset($_SESSION['informations']['nom'])?
                 $_SESSION['informations']['nom']:'',
     '<br />';
?>
```

*Résultat sur la page 2 après affichage de la page 1 et clic sur le lien*

```
$_SESSION['prénom'] = Olivier
$_SESSION['informations']['nom'] = HEURTEL
```

 Il n'y a aucun encodage (ni même de "magic quotes") sur les données enregistrées dans la session.

Le tableau $_SESSION est une variable "super-globale" : elle est disponible dans la totalité du script, même à l'intérieur des fonctions, sans devoir la déclarer globale (global $_SESSION est inutile).

Comme le montre l'exemple précédent, la fonction isset peut être utilisée pour tester si une donnée est enregistrée dans la session.

Si vous souhaitez supprimer une données de session, vous pouvez utiliser la fonction unset pour supprimer l'entrée dans le tableau $_SESSION : unset($_SESSION['prénom']).

Pour supprimer d'un seul coup toutes les données de session, vous pouvez affecter un tableau vide (array()) au tableau $_SESSION.

 Dans les deux cas, ne supprimez pas la totalité de la variable $_SESSION (unset($_SESSION)) !

## session_destroy

*Syntaxe*

```
booléen session_destroy()
```

Après un appel à la fonction session_destroy, la session n'existe plus ; un appel ultérieur à la fonction session_start va ouvrir une nouvelle session.

La fonction session_destroy retourne TRUE en cas de succès et FALSE en cas d'échec.

La fonction session_destroy échoue (et donc retourne FALSE) et affiche une alerte si elle est appelée avant la fonction session_start.

La fonction session_destroy ne supprime pas les données de session dans le script en cours. Pour supprimer immédiatement toutes les données de session, vous pouvez affecter un tableau vide à $_SESSION.

De même, cette fonction ne détruit pas le cookie de session éventuellement utilisé pour propager l'identifiant de session. Pour supprimer le cookie de session, vous pouvez utiliser la fonction setcookie (cf. dans ce chapitre - Utiliser des cookies).

*Exemple*

- Script page1.php

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Enregistrer une information dans la session.
$_SESSION['nom'] = 'Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <b>Page 1</b><br />
    <?php
    echo 'Bonjour ',$_SESSION['nom'],'<br />';
    echo 'session_id() = ',session_id(),'<br />';
    ?>
    <a href="page2.php">Page 2</a><br />
    </div>
  </body>
</html>
```

- Script page2.php

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Détruire la session.
session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 2</title></head>
  <body>
    <div>
    <b>Page 2</b><br />
    <?php
    echo 'Bonjour ',$_SESSION['nom'],'<br />';
    echo 'session_id() = ',session_id(),'<br />';
    ?>
    <a href="page3.php">Page 3</a><br />
    </div>
  </body>
</html>
```

- Script page3.php

```
<?php
// Ouvrir/réactiver la session.
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 3</title></head>
  <body>
    <div>
    <b>Page 3</b><br />
    <?php
    echo 'Bonjour ',$_SESSION['nom'],'<br />';
    echo 'session_id() = ',session_id(),'<br />';
    ?>
```

```
    </div>
  </body>
</html>
```

*Résultat*

- Page 1

**Page 1**
Bonjour Olivier
session_id() = defbd28027131043572a86e1bf41299b
Page 2

- Page 2

**Page 2**
Bonjour Olivier
session_id() =
Page 3

- Page 3

**Page 3**
Bonjour
session_id() = defbd28027131043572a86e1bf41299b

Dans la deuxième page, la donnée de session n'est pas encore supprimée. Par contre, dans la troisième page, la donnée est vide. C'est en fait une nouvelle session, mais le même identifiant a été réutilisé car le cookie de session n'avait par été supprimé.

Il est possible de modifier le script `page2.php` pour qu'il détruise complètement la session.

*Exemple*

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Effacer toutes les informations de session.
$_SESSION = array();
// Supprimer le cookie de session (si utilisé).
// Le cookie porte le nom de la variable qui stocke
// l'identifiant de session.
if (isset($_COOKIE[session_name()])) {
  setcookie(session_name(),'',time()-1,'/');
}
// Détruire la session.
session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 2</title></head>
  <body>
    <div>
    <b>Page 2</b><br />
    <?php
    echo 'Bonjour ',$_SESSION['nom'],'<br />';
    echo 'session_id() = ',session_id(),'<br />';
    ?>
    <a href="page3.php">Page 3</a><br />
    </div>
  </body>
</html>
```

*Résultat*

- Page 1

```
Page 1
Bonjour Olivier
session_id() = defbd28027131043572a86e1bf41299b
Page 2
```

- Page 2

```
Page 2
Bonjour
session_id() =
Page 3
```

- Page 3

```
Page 3
Bonjour
session_id() = 99f7fc425b98e6ff51d28c6da8502fc4
```

La donnée de session est bien effacée dès la deuxième page, et un nouvel identifiant de session est utilisé sur la troisième page.

## 3. Gérer soi-même la transmission de l'identifiant de session

### a. Description du problème

Normalement, l'identifiant de session est automatiquement transmis par PHP, soit par cookie soit par l'URL (si l'utilisateur n'accepte pas les cookies).

Néanmoins, trois directives de configuration pilotent ce comportement :

- `session.use_cookie`
- `session.use_trans_id`
- `session.use_only_cookie`

Si la directive `session.use_cookie` est égale à 0, PHP ne tente même pas d'utiliser les cookies pour transmettre l'identifiant de session. Par contre, si la directive est égale à 1 (valeur par défaut), PHP tente d'utiliser les cookies.

Si la directive `session.use_trans_sid` est égale à 0 (valeur par défaut), PHP n'utilise pas l'URL pour transmettre l'identifiant de session. Par contre, si la directive est égale à 1 et que l'identifiant de session ne puisse pas être transmis par cookie (du fait de la configuration ou d'un refus de l'utilisateur), alors PHP utilise l'URL pour transmettre l'identifiant de session.

Si la directive `session.use_only_cookie` est à 1, seuls les cookies sont utilisés pour transmettre l'identifiant de session. Par défaut, la directive est à 0 (compatibilité ascendante).

La conséquence principale est la suivante : si la directive `session.use_trans_sid` est égale à 0 (valeur par défaut) et si l'identifiant de session ne peut pas être transmis par cookie (du fait de la configuration ou d'un refus de l'utilisateur), la gestion des sessions ne fonctionne plus.

En terme de sécurité, il est déconseillé de permettre la transmission de l'identifiant de session par l'URL. Il est préférable d'utiliser des cookies, mais cela nécessite que l'internaute les accepte. Les valeurs par défaut des directives de configuration vont dans ce sens.

*Exemple*

- Script `page1.php`

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Récupérer l'identifiant de session.
$session = session_id();;
```

```
// Enregistrer une information dans la session.
$_SESSION['nom'] = 'Olivier';
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <b>Page 1</b><br />
    <?php
    // Afficher l'ID de la session.
    echo 'session_id() = ',session_id(),'<br />';
    // Afficher la donnée de session.
    echo 'nom = ',
         isset($_SESSION['nom'])?$_SESSION['nom']:'','<br />';
    ?>
    <a href="page2.php">Page 2</a><br />
    </div>
  </body>
</html>
```

- Script page2.php

```
<?php
// Ouvrir/réactiver la session.
session_start();?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 2</title></head>
  <body>
    <div>
    <b>Page 2</b><br />
    <?php
    // Afficher l'ID de la session.
    echo 'session_id() = ',session_id(),'<br />';
    // Afficher la donnée de session.
    echo 'nom = ',
         isset($_SESSION['nom'])?$_SESSION['nom']:'','<br />';
    ?>
    </div>
  </body>
</html>
```

*Résultat (premier cas)*

- Affichage initial de la page 1

**Page 1**
session_id() = 7580071d734e76516999f37d00a96cb7
nom = Olivier
Page 2

- Affichage de la page 2 après un clic sur le lien Page 2 (si l'utilisateur refuse les cookies et que session.use_trans_sid = 0)

**Page 2**
session_id() = 161492b7dd7f5738b9c8105bedf3250b
nom =

Les informations de session n'ont pas été transmises et l'appel à la fonction session_start a ouvert une nouvelle session.

*Résultat (deuxième cas)*

- Affichage initial de la page 1

```
Page 1
session_id() = 0283b72fb608b8255874c4e9144f8d47
nom = Olivier
Page 2
```

- Affichage de la page 1 après un clic sur le lien Page 2 (si l'utilisateur refuse les cookies et que `session.use_trans_sid = 1`)

```
Page 2
session_id() = 0283b72fb608b8255874c4e9144f8d47
nom = Olivier
```

Dans ce cas, la session a bien été conservée.

Regardons le code source HMTL de la première page pour comprendre ce qui s'est passé :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <div>
    <b>Page 1</b><br />
    session_id() = 0283b72fb608b8255874c4e9144f8d47<br />nom
= Olivier<br />    <a
href="page2.php?PHPSESSID=0283b72fb608b8255874c4e9144f8d47">
Page 2</a><br />
    </div>
  </body>
</html>
```

La balise `<a href>` a été automatiquement réécrite par PHP, pour intégrer la transmission de l'identifiant de session (nommé `PHPSESSID` par défaut). Si un paramètre était déjà présent dans l'URL, PHP aurait ajouté le paramètre `PHPSESSID` dans l'URL (`&PHPSESSID=...`).

Par contre, un problème se pose en cas de retour sur la page 1 par le bouton "Précédente" du navigateur :

```
Page 1
session_id() = 6be4fb067efdb037c4c6bb63fa0b5536
nom = Olivier
Page 2
```

Comme l'identifiant de session n'est pas retransmis lors de l'appel à cette page, une nouvelle session est ouverte.

De la même manière, si une page comporte un formulaire, une zone cachée est automatiquement ajoutée par PHP pour transmettre l'identifiant de session lors de la soumission du formulaire.

*Exemple*

```
<input type="hidden" name="PHPSESSID" value="556e46fb45de467e0b2a18ce46855016" />
```

Par contre, l'information n'est pas transmise lors d'une redirection avec la fonction `header`.

### b. Solution

Si l'identifiant n'est pas automatiquement transmis par PHP, il convient d'assurer soi-même cette transmission, le plus simple étant de le faire par l'URL (comme PHP le ferait si l'option était activée).

Une première possibilité consiste à utiliser les fonctions `session_name` et `session_id` pour construire le paramètre à insérer dans l'URL sous la forme `nom_identifiant=valeur_identifiant`.

La deuxième possibilité (plus simple), consiste à utiliser une constante, nommée `SID`, qui est automatiquement initialisée par PHP avec une chaîne `nom_identifiant=valeur_identifiant`.

*Exemple*

```
<?php
```

```
// Ouvrir/réactiver la session.
session_start();
// Construire la chaîne soi même.
echo 'Mon SID : ',session_name(),'=',session_id(),'<br />';
// Afficher la constante SID.
echo 'SID : ',SID;
?>
```

*Résultat*

```
Mon SID : PHPSESSID=8e7ebe7660d0441de4337d64bbec409a
SID : PHPSESSID=8e7ebe7660d0441de4337d64bbec409a
```

Dans les deux cas, si le nom de la variable a été modifié dans le fichier `php.ini`, les deux constructions en tiennent compte.

La constante `SID`, telle qu'elle est définie, peut être utilisée directement pour construire une URL transmettant la valeur de l'identifiant de session.

*Exemples*

- Dans une balise `<a href=...>` :

```
<a href="page2.php?<?php echo SID; ?>">Page 2</a>
```

- Dans l'attribut action de la balise `<form>` :

```
<form action="page2.php?<?php echo SID; ?>" method="post">
```

- Dans une redirection avec la fonction `header` :

```
header('Location: page2.php?'.SID);
```

Par contre, dans une zone cachée d'un formulaire, il faut utiliser les fonctions `session_name` et `session_id`.

*Exemple*

```
<input type = "hidden"
 name="<?php echo session_name(); ?>"
 value="<?php echo session_id(); ?>" />
```

---

La constante `SID` n'est pas renseignée si le cookie est accepté, sauf dans le script qui vient juste d'ouvrir la session (car PHP ne sait pas encore si le cookie est accepté par le client). Intégrer un `SID` vide dans une URL n'est pas très élégant mais ne pose pas de problème (`<a href="page2.php?">` est valide). Si vous intégrez systématiquement le `SID` dans les URL et que la transmission automatique fonctionne (soit par cookie, soit par l'URL), ce n'est pas non plus très élégant mais cela ne pose pas de problème.

---

Pour écrire du code portable et élégant, il convient donc de n'intégrer soi-même le `SID` dans l'URL que lorsque c'est nécessaire.

Une fonction générique peut très bien se charger de cette tâche en testant la directive de configuration et la constante `SID`.

*Exemple*

```
<?php
function url($url) {
// Si la directive de configuration session.use_trans_sid
// est à 0 (pas de transmission automatique par l'URL) et
// si SID est non vide (le poste a refusé le cookie) alors
// il faut gérer soi même la transmission.
if ((get_cfg_var('session.use_trans_sid') == 0)
    and (SID != '')) {
   // Ajouter la constante SID derrière l'URL avec un ?
   // s'il n'y a pas encore de paramètre, ou avec un & dans
   // le cas contraire.
```

```
    $url .= ((strpos($url,'?') === FALSE)?'?':'&').SID;
    }
    return $url;
}
// Ouvrir/réactiver la session.
session_start();
// Quelques tests
echo url('page2.php'),'<br />';
echo url('page3.php?nom=Olivier'),'<br />';
?>
```

*Résultat*

- Si `session.trans_id = 0` et cookie refusé :

```
page2.php?PHPSESSID=dda5ca567f453851a717ff118040e29f
page3.php?nom=Olivier&PHPSESSID=dda5ca567f453851a717ff118040e29f
```

- Si `session.trans_id = 0` et cookie accepté (mais 1er appel) :

```
page2.php?PHPSESSID=66312cb7e9c351fd73fb74b0bbdacd76
page3.php?nom=Olivier&PHPSESSID=66312cb7e9c351fd73fb74b0bbdacd76
```

- Si `session.trans_id = 0` et cookie accepté (mais 2ème appel) :

```
page2.php
page3.php?nom=Olivier
```

- Si `session.trans_id = 1` que le cookie soit refusé ou accepté :

```
page2.php
page3.php?nom=Olivier
```

Dans le dernier cas, lorsque le cookie est refusé, c'est PHP qui ajoute automatiquement le `SID` dans l'URL (car la directive `session.trans_id est à on`).

Cette fonction peut ensuite être appelée partout où il faut construire une URL susceptible de transmettre l'identifiant de session.



Quelle que soit la configuration, cette fonction doit être utilisée lors d'une redirection avec la fonction `header` (car PHP ne réécrit jamais les URL appelées par cette fonction).

## 4. Quelques directives de configuration supplémentaires

En complément de celles déjà citées, il est bon de connaître les directives suivantes :

`session.save_path`

Répertoire dans lequel les fichiers temporaires contenant les informations de session sont enregistrés.

`session.auto_start`

Si elle est positionnée à 1, la fonction `session_start` est automatiquement appelée au début de chaque script (pour un code portable, il est préférable d'appeler explicitement la fonction `session_start`).

`session.cookie_lifetime`

Durée de vie des cookies déposés sur le poste de l'utilisateur. La valeur 0 proposée par défaut est bien adaptée pour un cookie de session.

`session.cache_limiter`

Détermine le comportement vis-à-vis du cache de toutes les pages concernées par la gestion des sessions envoyées

au navigateur. Valeurs possibles : nocache, private, public. Cette information est transmise dans l'en-tête de la réponse du serveur Web. nocache par défaut.

session.cookie_path

Chemin sur le serveur dans lequel le cookie de session est disponible (voir la fonction setcookie). / par défaut.

session.cookie_domain

Domaine auquel le cookie est renvoyé (voir la fonction setcookie). Vide par défaut.

session.cookie_secure

Indique si le cookie ne doit être transmis que via une connexion sécurisée (voir la fonction setcookie). Off par défaut.

session.cache_expire

Durée de vie en minutes des pages dans le cache. 180 par défaut. Sans effet si la valeur nocache est spécifiée dans session.cache_limiter.

## 5. Exemple d'application

Les principes présentés dans les points précédents peuvent être illustrés sur une gestion de session avec authentification des utilisateurs.

*Source*

- Script login.php pour l'authentification

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include('fonctions.inc');
// Fonction qui vérifie que l'identification saisie
// est correcte.
function utilisateur_existe($identifiant,$mot_de_passe) {
  // Connexion et sélection de la base de données
  $connexion = mysqli_connect('localhost','root');
  mysqli_select_db($connexion,'eni');
  // Définition et exécution d'une requête préparée
  $sql  = 'SELECT 1 FROM utilisateurs ';
  $sql .= 'WHERE identifiant = ? AND mot_de_passe = ?';
  $requête = mysqli_stmt_init($connexion);
  $ok = mysqli_stmt_prepare($requête,$sql);
  $ok = mysqli_stmt_bind_param
          ($requête,'ss',$identifiant,$mot_de_passe);
  $ok = mysqli_stmt_execute($requête);
  mysqli_stmt_bind_result($requête,$existe);
  $ok = mysqli_stmt_fetch($requête);
  mysqli_stmt_free_result($requête);
  // L'identification est bonne si la requête a retourné
  // une ligne (l'utilisateur existe et le mot de passe
  // est bon).
  // Si c'est le cas $existe contient 1, sinon elle est
  // vide. Il suffit de la retourner en tant que booléen.
  return (bool) $existe;
}
// Initialisation des variables.
$identifiant = '';
$mot_de_passe = '';
$message = '';
// Traitement du formulaire.
if (isset($_POST['connexion'])) {
   // Récupérer les information saisies.
   $identifiant = valeur_saisie($_POST['identifiant']);
   $mot_de_passe = valeur_saisie($_POST['mot_de_passe']);
   // Vérifier que l'utilisateur existe.
```

```
    if (utilisateur_existe($identifiant,$mot_de_passe)) {
       // L’utilisateur existe ...
       // Ouvrir une session et enregistrer les données
       // de session.
       session_start();
       $_SESSION[’date’] = date("\l\e d/m/Y à H:i:s");
       $_SESSION[’identifiant’] = $identifiant;
       // Puis rediriger l’utilisateur vers la page d’accueil
       // en appelant la fonction générique url() pour être
       // certain que l’identifiant de session est transmis
       // quelles que soient les conditions.
       header(’location: ’.url(’accueil.php’));
       exit;
    } else {
       // L’utilisateur n’existe pas ...
       // Afficher un message et proposer de
       // nouveau l’identification.
       $message  = ’Identification incorrecte. ’;
       $message .= ’Essayez de nouveau.’;
       // Laisser le formulaire s’afficher de nouveau ...
    }
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>MonSite.com - Identification</title></head>
  <body>
    <form action="login.php" method="post">
    <table border="0">
    <tr>
      <td align="right">Identifiant :</td>
      <td><input type="text" Name="identifiant" value=
         "<?php echo vers_formulaire($identifiant); ?>" /></td>
    </tr>
    <tr>
      <td align="right">Mot de passe :</td>
      <td><input type="password" Name="mot_de_passe" value=
         "<?php echo vers_formulaire($mot_de_passe); ?>" /></td>
    </tr>
    <tr>
      <td></td>
      <td align="right"><input type="submit" name="connexion"
                         value="Connexion" /></td>
    </tr>
    </table>
    <?php echo $message; ?>
    </form>
  </body>
</html>
```

- Script accueil.php pour la page d’accueil

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include(’fonctions.inc’);
// Ouvrir/réactiver la session.
session_start();
// Tester si la session est nouvelle (ouverte par
// l’appel session_start() ci-dessus) ou ancienne (ouverte
// par un appel antérieur à session_start()).
// Le mieux est de tester si une de nos données de session
// est déjà enregistrée.
if (! isset($_SESSION[’identifiant’])) {
  // Donnée "identifiant" pas encore enregistrée :
  // => l’utilisateur n’est pas connecté ;
  // => le rediriger vers la page de login.
  header(’location: login.php’);
  exit;
```

```
} else {
  // Donnée "identifiant"déjà enregistrée :
  // => l'utilisateur est connecté ;
  // => récupérer les données de session utilisées dans
  //    le script.
  $date = $_SESSION ['date'];
  $identifiant = $_SESSION['identifiant'];
  // Récupérer l'identifiant de la session (pour l'exemple).
  $session = session_id();
  // Préparer un message.
  $message = "Session : $session - $identifiant - $date";
}
// Détermination de la date et de l'heure actuelle (pas celle
// de l'ouverture de la session).
$actuel = 'Nous sommes le '.date('d/m/Y').
          ' ; il est '.date('H:i:s');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>MonSite.com - Accueil</title></head>
  <body>
    <div>
      <b>Accueil - <?php echo $actuel; ?></b><br />
      <?php echo $message; ?><br />
      <!-- Lien vers une autre page.
           Utiliser notre fonction générique url() pour être
           certain que l'identifiant de session est transmis
           quelles que soient les conditions.  -->
      <a href="<?php echo url("action.php"); ?>">Action</a>
    </div>
  </body>
</html>
```

Le premier appel de l'URL `http://.../accueil.php` provoque la redirection de l'utilisateur vers la page d'identification ; après une identification réussie, l'utilisateur revient sur la page d'accueil qui affiche les informations suivantes :

```
Accueil - Nous sommes le 08/02/2008 ; il est 13:51:40
Session : 64b1740cbf6422491cd5860d3fa33a4a - heurtel - le 08/02/2008 à
13:51:40 Action
```

SI vous actualisez le contenu de la page quelques instants plus tard, l'heure actuelle change mais pas l'heure de l'ouverture de la session :

```
Accueil - Nous sommes le 08/02/2008 ; il est 13:53:59
Session : 64b1740cbf6422491cd5860d3fa33a4a - heurtel - le 08/02/2008 à
13:51:40 Action
```

## 6. Remarque et conclusion

### Les utilisateurs malins ou mal intentionnés

Le passage de l'identifiant de session par l'URL peut poser des problèmes de sécurité.

Considérons le script `page.php` suivant :

```
<?php
// Ouvrir/réactiver la session.
session_start();
// Afficher l'id de session.
echo 'session_id = ',session_id();
?>
```

Si l'URL `http://.../page.php?PHPSESSID=abc` est appelée, et si la directive `session.use_only_cookie` est à 0, nous obtenons le résultat suivant :

```
session_id : abc
```

La valeur donnée à la variable `PHPSESSID` dans l'URL a été prise comme identifiant de session. Cela peut permettre à un utilisateur (malin et/ou mal intentionné) de faire croire à votre application qu'il a déjà une session ouverte, ou d'utiliser une session ouverte par un autre utilisateur.

Ce phénomène se produit même si la directive `session.use_trans_id` est à 0, ce qui interdit à PHP de transmettre l'identifiant de session dans l'URL, mais pas de le recevoir.

Pour empêcher ce phénomène, il faut mettre la directive de configuration `session.use_only_cookies` à 1, afin de forcer l'utilisation d'un cookie pour la transmission de l'identifiant de session. L'inconvénient est que l'utilisateur peut refuser les cookies et que cette méthode n'est, de toute façon, pas complètement sécurisée ; la seule solution vraiment sécurisée consiste à utiliser une connexion sécurisée.

Indépendamment de cela, il est relativement simple de se prémunir contre l'utilisation d'un faux identifiant de session, soit en mémorisant côté serveur les identifiants des sessions ouvertes, soit en testant l'existence d'une donnée de session dans `$_SESSION`.

*Exemple*

```
<?php
// Ouvrir/réactiver la session.
session_start();
if (! isset($_SESSION['identifiant'])) {
  // Si la donnée de session 'identifiant'
  // n'existe pas, c'est que la session n'a
  // pas été réellement ouverte par l'application.
  // Faire ce qu'il faut ...
  // Pour cet exemple :
  // - afficher un message
  echo 'Session non ouverte','<br />';
  // - simuler l'ouverture applicative de la session
  $_SESSION['identifiant'] = '123';
} else {
  // Si la donnée de session 'identifiant' existe,
  // c'est que la session a réellement été ouverte
  // par l'application.
  echo 'Session ouverte','<br />';
  echo 'identifiant = ',$_SESSION['identifiant'],'<br />';
}
// Afficher l'id de session
echo 'session_id = ',session_id(),'<br />';
?>
```

*Résultat :*

- Premier appel du type `http://.../page.php?PHPSESSID=abc` :

```
Session non ouverte
session_id = abc
```

- Deuxième appel (sans fermer le navigateur) :

```
Session ouverte
identifiant = 123
session_id = abc
```

PHP a conservé l'identifiant de session initial. En cas de besoin, la fonction `session_regenerate_id` peut être appelée pour régénérer un identifiant de session.

*Exemple :*

```
<?php
// Ouvrir/réactiver la session.
session_start();
if (! isset($_SESSION['identifiant'])) {
  // Si la donnée de session 'identifiant'
  // n'existe pas, c'est que la session n'a
  // pas été réellement ouverte par l'application.
```

```
  // Faire ce qu'il faut ...
  // Pour cet exemple :
  // - régénérer un identifiant de session
  session_regenerate_id();
  // - afficher un message
  echo 'Session non ouverte','<br />';
  // - simuler l'ouverture applicative de la session
  $_SESSION['identifiant'] = '123';
} else {
  // Si la donnée de session 'identifiant' existe,
  // c'est que la session a réellement été ouverte
  // par l'application.
  echo 'Session ouverte','<br />';
  echo 'identifiant = ',$_SESSION['identifiant'],'<br />';
}
// Afficher l'id de session
echo 'session_id = ',session_id(),'<br />';
?>
```

_Résultat :_

- Premier appel du type `http://.../page.php?PHPSESSID=abc` :

```
Session non ouverte
session_id = 1863f3b9f496b9923e8b90fb2cb6b96b
```

- Deuxième appel (sans fermer le navigateur) :

```
Session ouverte
identifiant = 123
session_id = 1863f3b9f496b9923e8b90fb2cb6b96b
```

Cette technique ne met pas à l'abri de l'utilisation abusive par un internaute d'un identifiant de session d'un autre internaute.

_Résultat :_

- Un premier utilisateur effectue un appel du type `http://.../page.php` :

```
Session non ouverte
session_id = a5e762abc38f20f6a431ae81544fd080
```

- Un deuxième utilisateur réussit à récupérer l'identifiant de session du premier utilisateur et effectue un appel du type `http://.../page.php?PHPSESSID= a5e762abc38f20f6a431ae81544fd080`

```
Session ouverte
identifiant = 123
session_id = a5e762abc38f20f6a431ae81544fd080
```

 Si vous avez besoin d'un haut niveau de sécurité, utilisez une connexion sécurisée.

_Conclusion_

Moyennant un peu de rigueur dans le code, la fonctionnalité de gestion des sessions de PHP est facile à mettre en œuvre et indépendante des techniques de navigation utilisées (liens, formulaires). Pour des besoins de sécurité avancée, elle peut être facilement utilisée dans le cadre d'une connexion sécurisée ; dans ce cas, il s'agit surtout d'une problématique de configuration du serveur web et il n'y a rien de particulier à faire au niveau de PHP.

# Conserver des informations d’une visite à une autre

Si vous souhaitez pouvoir conserver des informations sur un utilisateur d’une visite à l’autre (éventuellement très éloignées dans le temps), il existe deux solutions prédominantes :

- déposer un cookie sur son poste (si possible avec son accord préalable) ;
- stocker les informations côté serveur (le plus pratique étant d’utiliser une base de données), et associer ces informations à une identification (typiquement un nom et un mot de passe) que l’utilisateur devra saisir à chaque visite.

Une solution intermédiaire, élégante et respectueuse de l’utilisateur est utilisée par certains sites ; cette solution consiste à proposer à l’utilisateur de déposer sur son poste un cookie qui ne contient qu’une ou deux informations permettant la connexion automatique au site (sans saisir de nom et de mot de passe), les informations complémentaires étant récupérées dans une base de données.

Nous allons illustrer cette solution à l’aide de deux pages :

- une page de personnalisation (script `personnaliser.php`) qui permet à l’utilisateur d’activer ou de désactiver la connexion automatique ;
- une page d’identification (script `login.php`) qui, selon le cas, effectue la connexion automatique ou demande à l’utilisateur de se connecter.

Chaque connexion de l’utilisateur est une session.

*Source*

- Script `personnaliser.php` :

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include('fonctions.inc');
// Ouvrir/réactiver la session.
session_start();
// Initialisation des variables.
$message = '';
// La session a-t-elle été ouverte au niveau applicatif ?
if (isset($_SESSION['identifiant'])) { // oui
  // Récupérer les informations de session.
  $identifiant = $_SESSION['identifiant'];
  $mot_de_passe = $_SESSION['mot_de_passe'];
  // Le script est-il appelé en traitement du formulaire ?
  if (isset($_POST['activer'])) { // oui
    // Activer la connexion automatique.
    // Déposer deux cookies d'une durée de vie de 30 jours,
    // un pour l'identifiant de l'utilisateur et un pour son
    // mot de passe.
    $expiration = time()+ (30 * 24 * 3600);
    setcookie('identifiant',$identifiant,$expiration);
    setcookie('mot_de_passe',$mot_de_passe,$expiration);
    // Préparer un message.
    $message = 'Connexion automatique activée';
  } elseif (isset($_POST['désactiver'])) { // oui
    // Désactiver la connexion automatique.
    // Supprimer les deux cookies.
    setcookie('identifiant');
    setcookie('mot_de_passe');
    // Préparer un message.
    $message = 'Connexion automatique désactivée';
  }
} else { // Session non ouverte au niveau applicatif.
  // Rediriger l'utilisateur vers la page de login
  header('Location: login.php');
  exit;
```

```
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>MonSite.com - Personnaliser</title></head>
  <body>
    <form action="personnaliser.php" method="post">
    <div>
    <input type="submit" name="activer"
           value="Activer la connexion automatique" /><br />
    <input type="submit" name="désactiver"
           value="Désactiver la connexion automatique" /><br />
    <?php echo $message; ?><br />
    </div>
    </form>
  </body>
</html>
```

- Script login.php (variante simple de la version précédente) :

```
<?php
// Inclusion du fichier contenant les fonctions générales.
include('fonctions.inc');
// Fonction qui vérifie que l'identification saisie
// est correcte.
function utilisateur_existe($identifiant,$mot_de_passe) {
  // Connexion et sélection de la base de données
  $connexion = mysqli_connect('localhost','root');
  mysqli_select_db($connexion,'eni');
  // Définition et exécution d'une requête préparée
  $sql  = 'SELECT 1 FROM utilisateurs ';
  $sql .= 'WHERE identifiant = ? AND mot_de_passe = ?';
  $requête = mysqli_stmt_init($connexion);
  $ok = mysqli_stmt_prepare($requête,$sql);
  $ok = mysqli_stmt_bind_param
          ($requête,'ss',$identifiant,$mot_de_passe);
  $ok = mysqli_stmt_execute($requête);
  mysqli_stmt_bind_result($requête,$existe);
  $ok = mysqli_stmt_fetch($requête);
  mysqli_stmt_free_result($requête);
  // L'identification est bonne si la requête a retourné
  // une ligne (l'utilisateur existe et le mot de passe
  // est bon).
  // Si c'est le cas $existe contient 1, sinon elle est
  // vide. Il suffit de la retourner en tant que booléen.
  return (bool) $existe;
}
// Initialisation des variables.
$identifiant = '';
$mot_de_passe = '';
$message = '';
$action = '';
// Le script est-il appelé en validation du formulaire ?
if (isset($_POST['connexion'])) { // oui
  // => connexion manuelle.
  // Récupérer les informations saisies.
  $identifiant = valeur_saisie($_POST['identifiant']);
  $mot_de_passe = valeur_saisie($_POST['mot_de_passe']);
  // Indiquer l'action à effectuer pour la suite.
  $action = 'connexion';
  // Préparer un message en cas de problème.
  $message = 'Identification incorrecte. '.
             'Essayez de nouveau.';
// Sinon, existe-t'il un cookie "identifiant" ?
} elseif (isset($_COOKIE['identifiant'])) { // oui
  // => connexion automatique.
  // Récupérer les informations des cookies
  $identifiant = valeur_saisie($_COOKIE['identifiant']);
```

```
  $mot_de_passe =  valeur_saisie($_COOKIE[’mot_de_passe’]);
  // Indiquer l’action à effectuer pour la suite.
  $action = ’connexion’;
  // Préparer un message en cas de problème.
  $message = ’Identification automatique incorrecte. ’.
             ’Essayez manuellement.’;
}
// Finalement, que fait-on ?
if ($action == ’connexion’) { // tenter une connexion
  // Vérifier que l’utilisateur existe.
  if (utilisateur_existe($identifiant,$mot_de_passe)) {
    // L’utilisateur existe ...
    // => ouvrir la session au niveau applicatif
    session_start();
    session_regenerate_id(); // au cas ou ...
    $_SESSION[’identifiant’] = $identifiant;
    $_SESSION[’mot_de_passe’] = $mot_de_passe;
    // Rediriger l’utilisateur vers une autre page du site
    // (il n’y en a qu’une !).
    header(’location: ’.url(’personnaliser.php’));
    exit;
  } // utilisateur_existe
} // $action == ’connexion’
// Si c’est le premier appel, ou si la connexion manuelle
// ou automatique a échoué, laisser le formulaire s’afficher.
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>MonSite.com - Identification</title></head>
  <body>
    <form action="login.php" method="post">
    <table border="0">
    <tr>
      <td align="right">Identifiant :</td>
      <td><input type="text" Name="identifiant" value=
           "<?php echo vers_formulaire($identifiant); ?>" /></td>
    </tr>
    <tr>
      <td align="right">Mot de passe :</td>
      <td><input type="password" Name="mot_de_passe" value=
           "<?php echo vers_formulaire($mot_de_passe); ?>" /></td>
    </tr>
    <tr>
      <td></td>
      <td align="right"><input type="submit" name="connexion"
                         value="Connexion" /></td>
    </tr>
    </table>
    <?php echo $message; ?>
    </form>
  </body>
</html>
```

*Résultat*

- Premier appel :

Identifiant :

Mot de passe :

Connexion

- Connexion manuelle erronée :

Identifiant : heurtel
Mot de passe : ****
Connexion
Identification incorrecte. Essayez de nouveau.

- Après une connexion réussie :

Activer la connexion automatique
Désactiver la connexion automatique

- Clic sur le bouton **Activer** :

Activer la connexion automatique
Désactiver la connexion automatique
Connexion automatique activée

- Quitter puis revenir sur une des deux pages. Arrivée directe sur la page de personnalisation (une connexion automatique s'est effectuée) :

Activer la connexion automatique
Désactiver la connexion automatique

- Clic sur le bouton **Désactiver** :

Activer la connexion automatique
Désactiver la connexion automatique
Connexion automatique désactivée

- Quitter puis revenir sur une des deux pages. La page d'identification est proposée (plus de connexion automatique) :

Identifiant :
Mot de passe :
Connexion

---

Il est préférable de stocker le mot de passe de l'utilisateur sous forme cryptée ou d'utiliser un identifiant de connexion automatique associé à l'utilisateur. De cette manière, dans les deux cas, le mot de passe de l'utilisateur n'est pas exposé aux regards indiscrets.

---

# Petite synthèse sur les variables GPCS (Get/Post/Cookie/Session)

Depuis le début de cet ouvrage, nous avons rencontré des variables "particulières", celles associées à des données de formulaire, à des données transmises par une URL, à des données d'un cookie ou encore, à des données de sessions.

Ces différents types de variables sont désignés sous le terme de variables GPCS (*Get/Post/Cookie/Session*).

Nous avons vu que ces variables fonctionnaient selon les mêmes principes :

- Elles sont accessibles directement sous la forme `$x` si la directive de configuration `register_globals` est à `on`.
- Elles sont aussi accessibles par l'intermédiaire de tableaux associatifs `$_GET`, `$_POST`, `$_COOKIE` et `$_SESSION`. En complément, le tableau associatif `$_REQUEST` regroupe le contenu des tableaux `$_GET`, `$_POST` et `$_COOKIE`.
- Les variables *Get/Post/Cookie* peuvent aussi être explicitement importées dans le script grâce à la fonction `import_request_variables`.

Pour chaque type, le conseil est le même : utilisez de préférence l'accès par le tableau associatif correspondant. Deux raisons justifient ce conseil :

- C'est la seule manière d'être certain que l'information qui est lue arrive bien par le moyen attendu.
- La directive de configuration `register_globals` est à `off` par défaut et passera définitivement à `off` dans une prochaine version de PHP.

Le tableau `$_REQUEST` doit être utilisé avec précaution car il contient des données fournies au script par plusieurs mécanismes ; nous ne sommes donc pas forcément certain que l'information lue arrive bien par le moyen attendu.

Le fait que les tableaux `$_GET`, `$_POST`, `$_COOKIE` et `$_SESSION` soient effectivement créés dépend de la directive de configuration `variables_order`.

Cette directive est une chaîne composée des caractères `G`, `P`, `C` et `S` correspondant aux types déjà évoqués et d'un cinquième caractère, `E`, correspondant aux variables d'environnement.

Les variables d'environnement correspondent aux variables d'environnement du système d'exploitation qui sont rendues disponibles dans l'environnement PHP, soit directement sous la forme `$x` (si la directive de configuration `register_globals` est à `on`) soit à travers le tableau associatif `$_ENV`.

Lorsque la directive de configuration `register_globals` est à `on`, l'ordre d'apparition des lettres `EGPCS` dans la directive `variables_order` correspond à un ordre de priorité croissant dans la création des variables.

Le contenu du tableau `$_REQUEST` est aussi conditionné par la directive `variables_order`.

Par défaut, la directive `variables_order` est égale à `EGPCS`.

*Exemple*

- Script `page1.php` :

```
<?php
// Ouvrir une session et enregistrer une information de
// session nommée "x" de valeur "SESSION".
session_start();
$_SESSION['x'] = 'SESSION';
// Déposer un cookie nommé "x" de valeur "COOKIE".
setcookie('x','COOKIE');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Page 1</title></head>
  <body>
    <!-- Dans un formulaire, mettre une variable donnée
         nommée "x" de valeur "GET" dans l'URL de
         l'attribut "action" -->
    <form action="page2.php?x=GET" method="post">
    <!-- Mettre aussi une zone nommée "x" de
```

```
          valeur "POST" -->
     <input type="hidden" name="x" value="POST" />
     <!-- Plus un bouton pour aller sur la page 2 -->
     <input type="submit" name="ok" value="Page 2">
     </form>
   </body>
</html>
```

- Script `page2.php` :

```
<?php
// Réactiver la session.
session_start();
// Afficher les valeurs de 'x' à partir des tableaux.
echo '$_GET[\'x\'] = ',
     isset($_GET['x'])?$_GET['x']:'','<br />';
echo '$_POST[\'x\'] = ',
     isset($_POST['x'])?$_POST['x']:'','<br />';
echo '$_COOKIE[\'x\'] = ',
     isset($_COOKIE['x'])?$_COOKIE['x']:'','<br />';
echo '$_SESSION[\'x\'] = ',
     isset($_SESSION['x'])?$_SESSION['x']:'','<br />';
echo '$_REQUEST[\'x\'] = ',
     isset($_REQUEST['x'])?$_REQUEST['x']:'','<br />';
?>
```

Dans le premier script, plusieurs informations différentes sont associées au même identifiant "x".

*Résultat de l'affichage de la page 1 puis du clic sur le bouton Page 2*

```
$_GET['x'] = GET
$_POST['x'] = POST
$_COOKIE['x'] = COOKIE
$_SESSION['x'] = SESSION
$_REQUEST['x'] = COOKIE
```

Le tableau `$_REQUEST` contient la valeur "COOKIE", car, dans la directive `variables_order`, la lettre `C` apparaît après les lettres `G` et `P` : l'information en provenance du cookie est donc prioritaire par rapport aux autres (les informations de session n'apparaissent pas dans le tableau `$_REQUEST`).

Si la directive `variables_order` est modifiée en `CP`, nous obtenons le résultat suivant :

```
$_GET['x'] =
$_POST['x'] = POST
$_COOKIE['x'] = COOKIE
$_SESSION['x'] = SESSION
$_REQUEST['x'] = POST
```

Le tableau `$_GET` n'est plus alimenté et le tableau `$_REQUEST` contient dorénavant la valeur "POST". Comme le montre cet exemple, le tableau `$_SESSION` est renseigné bien que la lettre `S` ne soit pas présente dans la directive `variables_order`.

Un problème similaire peut se produire lors de l'utilisation de la fonction `import_request_variables`. Le premier paramètre de cette fonction indique les types de variables à importer, sous la forme d'une combinaison des lettres `G`, `P` et `C` : `CP` par exemple. Les variables sont importées dans l'ordre indiqué et écrasent les variables de même nom précédemment importées. Sur notre exemple, la valeur "COOKIE" écrasera la valeur "POST". Pour éviter ce genre de problème, il est possible d'appeler plusieurs fois la fonction, en indiquant à chaque fois un préfixe différent pour le nom de la variable (deuxième paramètre).

---

Ce n'est pas une bonne habitude de programmation d'utiliser le même nom pour plusieurs informations ; un développeur sensé ne sera pas confronté à ce problème. Cette situation écartée, il faut avoir conscience qu'un utilisateur (malin et/ou mal intentionné) peut assez facilement fournir une valeur par un moyen donné (GPC en l'occurrence), à une variable dont vous pensez contrôler l'origine. Écrire du code dont le fonctionnement est lié à une certaine valeur de la directive `variables_order` n'est sans doute pas, non plus, une bonne idée du point de vue de la portabilité et de la maintenabilité.

---

Le conseil avisé, donné par l'équipe de développement de PHP, est donc le suivant :

---

Récupérez les valeurs "EGPCS" par les tableaux associatifs pour éviter tout problème.

---

C’est pour cela que la directive `register_globals` est à `off` par défaut, et qu’elle sera forcée définitivement à `off` dans une prochaine version de PHP.

# Variables PHP prédéfinies

PHP prédéfinit un grand nombre de variables relatives à son fonctionnement. Ces variables sont directement accessibles sous la forme `$x` si la directive de configuration `register_globals` est à `on`, mais ce n'est pas la méthode recommandée. Pour accéder à ces informations, il est préférable d'utiliser les tableaux associatifs proposés par PHP.

Les tableaux associatifs sont les suivants :

| Nom | Contenu |
|---|---|
| `$GLOBALS` | Tableau associatif de toutes les variables disponibles dans la portée du script (chapitre Ecrire des fonctions et des classes PHP). |
| `$_COOKIE` | Tableau associatif des variables passées au script par les cookies (chapitre Gérer les sessions). |
| `$_GET` | Tableau associatif des variables passées au script par une méthode GET (chapitres Gérer les formulaires et les liens avec PHP). |
| `$_POST` | Tableau associatif des variables passées au script par une méthode POST (chapitres Gérer les formulaires et les liens avec PHP) |
| `$_FILES` | Tableau associatif contenant les informations sur les fichiers téléchargés du poste de l'utilisateur vers le serveur Web (chapitre Gérer les formulaires et les liens avec PHP). |
| `$_ENV` | Tableau associatif des variables d'environnement du système d'exploitation passées au script (chapitre Gérer les sessions). |
| `$_SERVER` | Tableau associatif des variables du serveur passées au script (variable d'environnement et variables du serveur HTTP notamment). |
| `$_REQUEST` | Tableau associatif regroupant les tableaux `$_GET`, `$_POST` et `$_COOKIE` (chapitres Gérer les formulaires et les liens avec PHP et Gérer les sessions). |
| `$_SESSION` | Tableau associatif des données de session accessible dans le script (chapitre Gérer les sessions). |

Ces tableaux associatifs sont des variables "super globales" : ils sont disponibles dans la totalité du script, même à l'intérieur des fonctions, sans devoir les déclarer globaux (`global $...` est inutile).

*Exemple : affichage partiel du contenu de $_SERVER*

```
HTTP_HOST = xampp
HTTP_USER_AGENT = Mozilla/5.0 (Windows; U; Windows NT 5.1; fr; rv:
.8.1.11) Gecko/20071127 Firefox/2.0.0.11
HTTP_ACCEPT = text/xml,application/xml,application/xhtml
+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
HTTP_ACCEPT_LANGUAGE = fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
HTTP_ACCEPT_ENCODING = gzip,deflate
HTTP_ACCEPT_CHARSET = ISO-8859-1,utf-8;q=0.7,*;q=0.7
HTTP_KEEP_ALIVE = 300
HTTP_CONNECTION = keep-alive
HTTP_REFERER = http://xampp/eni/
CONTENT_TYPE = application/x-www-form-urlencoded
CONTENT_LENGTH = 97
SERVER_SOFTWARE = Apache/2.2.6 (Unix) DAV/2 mod_ssl/2.2.6 OpenSSL/0.9.8e
PHP/5.2.4 mod_apreq2-20051231/2.5.7 mod_perl/2.0.2 Perl/v5.8.7
```

```
SERVER_NAME = xampp
SERVER_ADDR = 192.168.154.200
SERVER_PORT = 80
REMOTE_ADDR = 192.168.154.1
DOCUMENT_ROOT = /opt/lampp/htdocs
SERVER_ADMIN = you@example.com
SCRIPT_FILENAME = /app/scripts/index.php
REMOTE_PORT = 42134
GATEWAY_INTERFACE = CGI/1.1
SERVER_PROTOCOL = HTTP/1.1
REQUEST_METHOD = POST
QUERY_STRING =
REQUEST_URI = /eni/
SCRIPT_NAME = /eni/index.php
PHP_SELF = /eni/index.php
REQUEST_TIME = 1202411345
```

Quelques informations intéressantes apparaissent en gras.

La fonction `phpinfo` permet d'afficher ces différentes informations (entre autres).

# Constantes PHP prédéfinies

PHP prédéfinit un grand nombre de constantes parmi lesquelles :

| Nom | Contenu |
|---|---|
| `__FILE__ *` | Nom du fichier en cours d'exécution. Donne le nom du fichier inclus si elle est utilisée dans un fichier inclus. |
| `__LINE__ *` | Numéro de la ligne actuellement exécutée. Donne le numéro de ligne dans le fichier inclus si elle est utilisée dans un fichier inclus. |
| `PHP_VERSION` | Version de PHP. |
| `PHP_OS` | Système d'exploitation du serveur PHP. Exemples (non exhaustifs) : `AIX`, `Linux`, `SunOS`, `WINNT`. |
| `TRUE *` | Valeur booléenne vraie (`TRUE`). |
| `FALSE *` | Valeur booléenne fausse (`FALSE`). |
| `E_*` | Codes des erreurs (chapitre Gérer les erreurs dans un script PHP). |
| `DIRECTORY_SEPARATOR` | Caractère de séparation utilisé dans les noms de répertoire pour la plate-forme sur laquelle PHP est installée. |
| `PHP_EOL` | Séquence de caractères utilisée par la plate-forme pour représenter une nouvelle ligne. Ajouté en version 5.0.2. |
| `PHP_INT_MAX` | Valeur du plus grand entier. Ajouté en version 5.0.5. |
| `PHP_INT_SIZE` | Taille des entiers (nombre d'octets). Ajouté en version 5.0.5. |

Les constantes suivies d'une étoile sont insensibles à la casse (et peuvent être utilisées indifféremment en majuscules ou en minuscules).

# Exemples complémentaires

## 1. Introduction

La puissance du langage PHP est renforcée par l'existence d'un grand nombre de bibliothèques qui étendent les fonctionnalités du langage : vérification d'orthographe, génération de documents PDF (*Portable Document Format*), manipulation de documents XML (*Extensible Markup Language*), accès à des serveurs FTP (*File Transfer Protocol*), accès à des serveurs IMAP (*Internet Message Access Protocol*), accès à des annuaires LDAP (*Lightweight Directory Access Protocol*), cryptage, génération de documents Shockwave Flash, gestion du protocole SNMP (*Simple Network Management Protocol*), compression, etc. Certaines bibliothèques nécessitent des librairies complémentaires. Ces différentes bibliothèques correspondent à des besoins particuliers et ne sont pas décrites plus en détail dans cet ouvrage.

Dans cette partie de l'annexe, nous allons donner trois exemples commentés d'utilisation de ces bibliothèques correspondant à trois besoins fréquents :

- lire un document XML ;
- générer un document PDF ;
- générer une image.

Pour plus d'informations sur les bibliothèques disponibles ou sur une fonction, reportez-vous à l'aide en ligne accessible sur le site officiel de PHP (www.php.net/manual/fr/).

## 2. Lire un document XML

Cet exemple illustre les possibilités de l'extension SimpleXML apparue en version 5.

Pour cet exemple, nous supposons qu'une liste d'articles est stockée dans un fichier nommé `articles.xml` :

```
<?xml version='1.0' encoding='UTF-8'?>
<articles>
     <article code="A1" couleur="jaune">
          <identifiant>1</identifiant>
          <libelle>Abricots</libelle>
          <prix>35.5</prix>
     </article>
     <article code="A2" couleur="rouge">
          <identifiant>2</identifiant>
          <libelle>Cerises</libelle>
          <prix>48.9</prix>
     </article>
     <article code="A3" couleur="rouge">
          <identifiant>3</identifiant>
          <libelle>Fraises</libelle>
          <prix>29.95</prix>
     </article>
     <article code="A4" couleur="jaune">
          <identifiant>4</identifiant>
          <libelle>Pêches</libelle>
     <prix>37.2</prix>
     </article>
</articles>
```

*Code*

```
<?php

// Charger le document XML = simplexml_load_file()
// - retourne un objet de la classe simplexml_element
//   ou FALSE en cas d'erreur (document XML mal formé par exemple)
$xml = simplexml_load_file('articles.xml');
if (! $xml) { exit; }
```

```
// L'objet $xml a une structure qui correspond à la
// structure de notre document :
// - article (tableau d'objets)
//     - identifiant
//     - libelle
//     - prix

// Parcours du noeud article (tableau).
echo "<b>Parcours de \$xml->article</b><br />\n";
foreach ($xml->article as $article) {
  printf("%s,%s,%s,%s,%s<br />\n",
         $article->identifiant,
         $article->libelle,
         $article->prix,
         $article['code'],
         $article['couleur']);
}

// Accès à une information particulière.
echo "<b>Accès à une information particulière</b><br />\n";
printf("Avant - Prix de %s = %s (code = %s)<br />\n",
       $xml->article[2]->libelle,
       $xml->article[2]->prix,
       $xml->article[2]['code']);
$xml->article[2]->prix = 123;
$xml->article[2]['code'] .= '+';
printf("Après - Prix de %s = %s (code = %s)<br />\n",
       $xml->article[2]->libelle,
       $xml->article[2]->prix,
       $xml->article[2]['code']);

// Extraction des attributs d'un noeud = méthode attributes()
// - retourne un objet de la classe simplexml_element
// - sur notre exemple, récupération des attributs du 1er article
$attributs = $xml->article[0]->attributes();

// Parcours des attributs ainsi récupérés.
echo "<b>Attributs du premier article</b><br />\n";
foreach($attributs as $nom => $valeur) {
  printf("%s = %s<br />\n",$nom,$valeur);
}

// Extraire les enfants d'un noeud = méthode children()
// - retourne un objet de la classe simplexml_element
echo "<b>Parcours de l'arborescence</b><br />\n";
echo "racine<br />\n";
foreach ($xml->children() as $nom1 => $niveau1) {
  printf("----%s (%s,%s)<br />\n",
         $nom1,$niveau1['code'],$niveau1['couleur']);
  foreach ($niveau1->children() as $nom2 => $niveau2) {
    printf("--------%s = %s<br />\n",$nom2,$niveau2);
  }
}

// Effectuer une recherche Xpath = méthode xpath()
// - retourne un tableau d'objets de la classe simplexml_element
echo "<b>Recherche Xpath : /articles/article</b><br />\n";
$résultat = $xml->xpath("/articles/article");
foreach ($résultat as $valeur) {
  printf("%s,%s<br />\n",$valeur->identifiant,$valeur->libelle);
}
echo "<b>Recherche Xpath : article/libelle</b><br />\n";
$résultat = $xml->xpath("article/libelle");
foreach ($résultat as $valeur) {
  printf("%s<br />\n",$valeur);
}
echo "<b>Recherche Xpath : //prix</b><br />\n";
$résultat = $xml->xpath("//prix");
```

```
foreach ($résultat as $valeur) {
  printf("%s<br />\n",$valeur);
}

// Générer une chaîne XML = méthode asXML()
echo "<b>Chaîne XML</b><br />\n";
file_put_contents ('les_articles.xml',$xml->asXML());
file_put_contents ('un_article.xml',$xml->article[0]->asXML());
echo "Voir les fichiers 'les_articles.xml' et 'un_article.xml'<br />\n";

?>
```

*Résultat à l'écran*

```
Parcours de $xml->article
1,Abricots,35.5,A1,jaune
2,Cerises,48.9,A2,rouge
3,Fraises,29.95,A3,rouge
4,Pêches,37.2,A4,jaune
Accès à une information particulière
Avant - Prix de Fraises = 29.95 (code = A3)
Après - Prix de Fraises = 123 (code = A3+)
Attributs du premier article
code = A1
couleur = jaune
Parcours de l'arborescence
racine
----article (A1,jaune)
--------identifiant = 1
--------libelle = Abricots
--------prix = 35.5
----article (A2,rouge)
--------identifiant = 2
--------libelle = Cerises
--------prix = 48.9
----article (A3+,rouge)
--------identifiant = 3
--------libelle = Fraises
--------prix = 123
----article (A4,jaune)
--------identifiant = 4
--------libelle = Pêches
--------prix = 37.2
Recherche Xpath : /articles/article
1,Abricots
2,Cerises
3,Fraises
4,Pêches
Recherche Xpath : article/libelle
Abricots
Cerises
Fraises
Pêches
Recherche Xpath : //prix
35.5
48.9
123
37.2
Chaîne XML
Voir les fichiers 'les_articles.xml' et 'un_article.xml'
```

*Contenu du fichier `les_articles.xml`*

```
<?xml version="1.0" encoding="UTF-8"?>
<articles>
     <article code="A1" couleur="jaune">
          <identifiant>1</identifiant>
          <libelle>Abricots</libelle>
```

```
            <prix>35.5</prix>
      </article>
      <article code="A2" couleur="rouge">
            <identifiant>2</identifiant>
            <libelle>Cerises</libelle>
            <prix>48.9</prix>
      </article>
      <article code="A3+" couleur="rouge">
            <identifiant>3</identifiant>
            <libelle>Fraises</libelle>
            <prix>123</prix>
      </article>
      <article code="A4" couleur="jaune">
            <identifiant>4</identifiant>
            <libelle>Pêches</libelle>
            <prix>37.2</prix>
      </article>
</articles>
```

*Contenu du fichier `un_article.xml`*

```
<article code="A1" couleur="jaune">
            <identifiant>1</identifiant>
            <libelle>Abricots</libelle>
            <prix>35.5</prix>
      </article>
```

## 3. Générer un document PDF

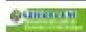
L'extension PDF de PHP permet de générer des documents PDF. Cette extension utilise la librairie PDFlib qui nécessite une licence (http://www.pdflib.com/products/pdflib-family/).

Il existe plusieurs alternatives gratuites pour générer des documents PDF, dont la librairie FPDF (http://www.fpdf.org/). Cette librairie n'est pas installée par défaut avec PHP mais elle est présente dans le package XAMPP. C'est cette librairie que nous allons utiliser dans cet exemple.

Le code ci-après permet de générer le document PDF suivant :

**Liste des collections**

| Nom | Prix H.T. |
|---|---|
| Ressources Informatiques | 24,44 |
| TechNote | 9,48 |
| Les TP Informatiques | 25,59 |
| Coffret Technique | 46,45 |



Page 1

La liste des collections est lue dans la base de données.

*Code*

```
<?php

// Inclusion de la librairie.
include ('fpdf.php');

// Sélectionner les données dans la base de données.
set_magic_quotes_runtime(0);
$db = @mysqli_connect('localhost','eniweb','web','eni');
if (! $db) {
  exit('Erreur lors de la connexion.');
}
$sql = 'SELECT nom,prix_ht FROM collection LIMIT 4';
$ok = ($requête = mysqli_prepare($db, $sql));
if ($ok) { $ok = @mysqli_stmt_execute($requête); }
if ($ok) { $ok = @mysqli_stmt_bind_result($requête,$nom,$prix); }
if (! $ok) {
  exit('Erreur lors de la sélection des données dans la base.');
}

// Créer un nouveau document PDF = new FPDF()
// - premier paramètre = orientation
```

```
//     > P = portrait - L = Paysage
// - deuxième paramètre = unité de mesure
//     > pt = point - mm = millimètre - cm = centimètre
// - troisième paramètre = format (A3, A4, etc)
// Tous les paramètres sont optionnels. Défaut = P, mm, A4.
$pdf = new FPDF('P','mm','A4');

// Définir les sauts de page automatiques = SetAutoPageBreak()
// - premier paramètre = automatique (true/false)
//     > P = portrait - L = Paysage
// - deuxième paramètre = marge
//     > distance par rapport au bas de la page qui déclenche
//       le saut (2 cm par défaut, si actif)
$pdf->SetAutoPageBreak(false);

// Créer une nouvelle page dans le document = AddPage()
// - premier paramètre = orientation
//     > P = portrait - L = Paysage
//       Par défaut, celle du document.
$pdf->AddPage();

// Définir les informations de résumé du document = SetTitle(),
// SetAuthor(), SetSubject().
$pdf->SetTitle('Liste des collections');
$pdf->SetAuthor('Olivier HEURTEL');
$pdf->SetSubject('Collections');

// Définir la police à utiliser = SetFont()
// - premier paramètre = famille
//     > nom d'une famille standard (Courier, Helvetica ou Arial,
//       Times, Symbol, ZapfDingbats) ou d'un nom défini par
//       AddFont().
// - deuxième paramètre (optionnel) = style
//     > combinaison de : B = gras - I = italique - U = souligné
// - troisième paramètre (optionnel) = taille en points
// Voir aussi la méthode SetFontSize() pour modifier la taille.
$pdf->SetFont('Arial','B',16);

// Ecrire du texte à partir de la position courante = Write()
// - premier paramètre = hauteur de la ligne
// - deuxième paramètre = texte à écrire
// Utilise les caractéristiques actuelles de police, couleurs, etc.
// Le retour à la ligne est automatique lorsque la marge droite est
// atteinte (ou que le caractère \n est rencontré).
$pdf->Write(5, 'Liste des collections');

// Effectuer un saut de ligne = Ln()
// - premier paramètre (optionnel) = hauteur de la ligne
// L'abcisse revient à la valeur de la marge gauche.
$pdf->Ln(10);

// Changer la taille de police = SetFontSize()
// - premier paramètre = taille en points
$pdf->SetFontSize(12);

// Définir la couleur à utiliser pour le texte = SetTextColor()
// - si un seul paramètre = niveau de gris (entre 0 et 255)
// - si 3 paramètres = composantes RGB (entre 0 et 255)
$pdf->SetTextColor(255,0,0); // rouge

// Définir la couleur à utiliser pour le fond = SetFillColor()
// - si un seul paramètre = niveau de gris (entre 0 et 255)
// - si 3 paramètres = composantes RGB (entre 0 et 255)
$pdf->SetFillColor(255,255,140); // jaune pale

// Ecrire une cellule = Cell()
// - premier paramètre = largeur (0 = jusqu'à la marge droite)
// - deuxième paramètre = hauteur
```

```
// - troisième paramètre = texte à écrire
// - quatrième paramètre = bordure
//     > soit un nombre  : 0 = aucun bord - 1 = cadre
//     > soit une chaîne : combinaison de L (gauche), T (haut),
//                         R (droit), B (bas)
// - cinquième paramètre = position à la fin
//     > 0 = à droite - 1 = début ligne suivante - 2 = en dessous
// - sixième paramètre = alignement
//     > L ou chaîne vide = à gauche - C = centré - R = à droite
// - septième paramètre = remplissage
//     > 0 = non - 1 = oui
// Seul le premier paramètre est obligatoire.
$pdf->Cell(80,7,'Nom',1,0,'C',1); // titre de colonne
$pdf->Cell(40,7,'Prix H.T.',1,1,'C',1); // titre de colonne

// Changer de couleur et de police
$pdf->SetFont('Arial','',12); // '' = normal
$pdf->SetTextColor(0,0,0); // noir

// Dans une boucle, écrire les données du tableau.
while (mysqli_stmt_fetch($requête)) { // fetch de la requête
  $prix = number_format($prix,2,',',' ');
  $pdf->Cell(80,7,$nom,1);
  $pdf->Cell(40,7,$prix,1,1,'R'); // ligne suivante + à droite
}

// Se positionner à un endroit précis dans la page = SetXY()
// - premier paramètre = abscisse (x)
// - deuxième paramètre = ordonnée (y)
// L'origine est le coin supérieur gauche.
// Si les valeurs sont négatives, l'origine est le coin
// inférieur droit.
// Voir aussi SetX() et SetY().
$pdf->SetXY(10,-10); // 1 cm à gauche, 1 cm du bas

// Afficher le numéro de page = PageNo()
$pdf->SetFontSize(10);
$pdf->Cell(0,0,'Page '.$pdf->PageNo(),0,0,'R');

// Afficher une image = Image()
// - premier paramètre = nom du fichier
// - deuxième paramètre = abscisse du coin supérieur gauche
// - troisième paramètre = ordonnée du coin supérieur gauche
// - quatrième paramètre (optionnel) = largeur de l'image
//     > 0 ou absent = calculée automatiquement
// - cinquième paramètre (optionnel) = hauteur de l'image
//     > 0 ou absent = calculée automatiquement
// - sixième paramètre (optionnel) = type
//     > JPG ou JPEG ou PNG
//     > Déduit de l'extension si absent
$pdf->Image('logo.jpg',10,285,20);

// Définir la couleur à utiliser pour le dessin = SetDrawColor()
// - si un seul paramètre = niveau de gris (entre 0 et 255)
// - si 3 paramètres = composantes RGB (entre 0 et 255)
$pdf->SetDrawColor(128); // niveau de gris
$pdf->line(10,15,200,15); // ligne horizontale en haut
$pdf->line(10,285-2,200,285-2); // ligne horizontale en bas

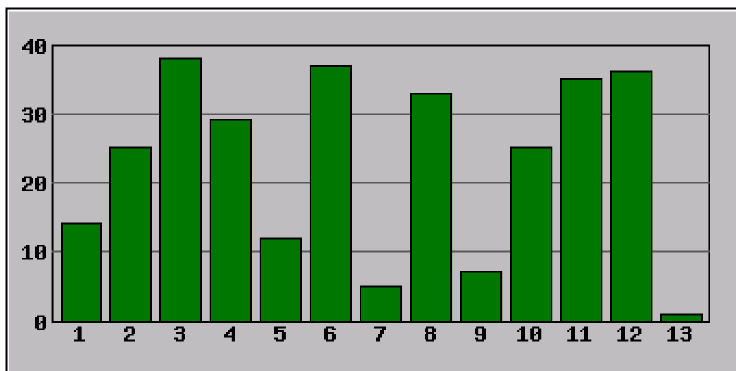
// Envoyer le document vers une destination = Output()
// - premier paramètre (optionnel) = nom du fichier
// - deuxième paramètre (optionnel) = type de destination
//     > F = fichier sur le serveur
//     > I = navigateur (en ligne)
//     > D = navigateur (téléchargement)
// Si aucun paramètre : destination = I
// Si un nom est spécifié : destination par défaut = F
$pdf->Output(); // navigateur (en ligne)
```

```
?>
```

## 4. Générer une image

L'extension GD de PHP permet de créer et manipuler des images.

Le code ci-dessous permet de générer la page suivante :




La génération du graphique est effectuée dynamiquement lors de l'appel à la page ; pour cet exemple, les données du graphique sont calculées aléatoirement.

*Code*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Mon beau graphique</title>
  </head>
  <body>
    <div>
    <b>Mon beau graphique</b>
    <p>
    <img alt="Mon beau graphique" src="mon-beau-graphique.php" />
    </p>
  </body>
</html>
```

Notez que la source de l'image est un script PHP qui crée l'image dynamiquement lorsque la page est affichée. Cette technique peut aussi être utilisée pour afficher dans une page une image qui est stockée dans une base de données.

*Script mon-beau-graphique.php*

```
<?php

// Définir un en-tête indiquant qu'il s'agit d'une image (ici PNG).
header('Content-type: image/png');

/*

Les fonctions définies dans le script utilisent deux variables globales :
- $image         = ressource image en cours de création
- $hauteur_image = hauteur de l'image

Pour les coordonnées, l'origine de l'image est le coin supérieur gauche.
Pour le dessin de notre graphique, nous utilisons une origine dans le
coin inférieur gauche (plus pratique). Les fonctions effectuent la
conversion.

*/
```

```
// Fontion de dessin d'un rectangle
// - $x1,$y1         = point 1
// - $x2,$y2         = point 2
// - $bordure, $fond = couleurs de la bordure et du fond
//
function rectangle($x1,$y1,$x2,$y2,$bordure,$fond) {

  global $image;
  global $hauteur_image;

  // Conversion du système de coordonnées pour l'axe y.
  $y1 = $hauteur_image - 1 - $y1;
  $y2 = $hauteur_image - 1 - $y2;

  // Dessiner le bord d'un rectangle = imagerectangle
  imagerectangle($image,$x1,$y1,$x2,$y2,$bordure);

  // Remplissage (si demandé i.e. $fond renseigné)
  if ( ( $fond != NULL) and ($x1 != $x2) and ($y1 != $y2) ) {

    // Remplir un rectangle = imagefilledrectangle
    imagefilledrectangle($image,$x1+1,$y1-1,$x2-1,$y2+1,$fond);
  }
}

// Fontion de dessin d'une ligne
// - $x1,$y1  = point 1
// - $x2,$y2  = point 2
// - $couleur = couleur
//
function ligne($x1,$y1,$x2,$y2,$couleur) {

  global $image;
  global $hauteur_image;

  // Conversion du système de coordonnées pour l'axe y
  $y1 = $hauteur_image - 1 - $y1;
  $y2 = $hauteur_image - 1 - $y2;

  // Dessiner une ligne = imageline
  imageline($image,$x1,$y1,$x2,$y2,$couleur);

}

// Fontion de dessin d'un texte
// - $police     = code de la police prédéfinie (1 à 5)
// - $x,$y       = point de référence
// - $texte      = texte
// - $couleur    = couleur
// - $horizontal = alignement horizontal par rapport au point
//                 de référence
//      > D = aligné à droite, C = centré, G = aligné à gauche
// - $vertical   = alignement vertical par rapport au point de référence
//      > H = texte en haut, C = centré, B = texte en bas
//
function texte($police,$x,$y,$texte,$couleur,$horizontal,$vertical) {

  global $image;
  global $hauteur_image;

  // Conversion du système de coordonnées pour l'axe y
  $y = $hauteur_image - 1 - $y;

  // Calculer la largeur d'un caractère dans une police =
  // imagefontwidth
  $largeur = imagefontwidth($police) * strlen($texte);

  // Calculer la hauteur d'un caractère dans une police =
  // imagefontheight
  $hauteur = imagefontheight($police);
```

```
  // Calcul des coordonnées en fonction de l'alignement
  switch ($horizontal) {
    case 'D':
      $x = $x - $largeur;
      break;
    case 'C':
      $x = $x - floor($largeur/2);
      break;
    case 'G':
      break;
  }
  switch ($vertical) {
    case 'H':
      $y = $y - $hauteur;
      break;
    case 'C':
      $y = $y - floor($hauteur/2);
      break;
    case 'B':
      break;
  }

  // Dessiner un texte = imagestring
  imagestring($image,$police,$x,$y,$texte,$couleur);
}

// Pour cet exemple, les données du graphe sont calculées de
// façon aléatoires.
// - unité, valeur min et valeur max pour l'axe y
$axe_y_unite = 10;
$axe_y_min = 0;
$axe_y_max = 40;
// - nombre de barres : entre 5 et 15
$nombre_barres = rand(5,15);
// - mettre les données dans le tableau
$données = array();
for ($i = 1 ; $i <= $nombre_barres ; $i++) {
  $données[$i] = rand($axe_y_min,$axe_y_max);
}

// Dimensions du dessin (pixels)
$largeur_image = 400; // largeur de l'image
$hauteur_image = 200; // hauteur de l'image
$marge_blanche = 2;   // marge blanche
$marge_gauche = 25;   // marge gauche avec la zone de traçage
$marge_droite = 20;   // marge droite avec la zone de traçage
$marge_haute = 20;    // marge haute avec la zone de traçage
$marge_basse = 30;    // marge basse avec la zone de traçage
$écart_barres = 5;    // écart entre les barres

// En déduire la largeur et la hauteur de la zone de traçage.
$largeur_tracé = $largeur_image - $marge_droite - $marge_gauche;
$hauteur_tracé = $hauteur_image - $marge_haute - $marge_basse;

// En déduire la largeur des barres et l'échelle de l'axe y.
$largeur_barre = ($largeur_tracé-$écart_barres)/$nombre_barres - $écart_barres ;
$échelle_axe_y = ($hauteur_tracé-1) / $axe_y_max ;

// Créer l'image = imagecreatetruecolor
$image = imagecreatetruecolor($largeur_image,$hauteur_image);

// Définir des couleurs = imagecolorallocate
// - en RGB
$blanc = imagecolorallocate($image, 255, 255, 255);
$noir = imagecolorallocate($image, 0, 0, 0);
$gris_clair = imagecolorallocate($image, 192, 192, 192);
$gris_foncé = imagecolorallocate($image, 100, 100, 100);
$vert = imagecolorallocate($image, 0, 128, 0);
```

```
// Coordonnées de l'image.
$ox1 = 0; $oy1 = 0;
$ox2 = $largeur_image - 1; $oy2 = $hauteur_image - 1;
// dessiner le cadre extérieur
rectangle($ox1,$oy1,$ox2,$oy2,$noir,$blanc);

// Coordonnées de l'image moins le bord blanc.
$mx1 = $ox1 + $marge_blanche; $my1 = $oy1 + $marge_blanche;
$mx2 = $ox2 - $marge_blanche; $my2 = $oy2 - $marge_blanche;
// dessiner le fond gris clair
rectangle($mx1,$my1,$mx2,$my2,$gris_clair,$gris_clair);

// Coordonnées de la zone de traçage.
$tx1 = $ox1 + $marge_gauche; $ty1 = $oy1 + $marge_basse;
$tx2 = $ox2 - $marge_droite; $ty2 = $oy2 - $marge_haute;
// dessiner le cadre de la zone de traçage
rectangle($tx1,$ty1,$tx2,$ty2,$noir,NULL);

// Dessin des lignes horizontales avec leur étiquette.
$x1 = $tx1;
$x2 = $tx2;
for ($axe = $axe_y_min ; $axe <= $axe_y_max ; $axe += $axe_y_unite) {
  $y1 = $ty1 + $axe * $échelle_axe_y;
  $y2 = $y1;
  // ne pas dessiner la ligne en bas et en haut
  if ( ( $axe > $axe_y_min ) and ( $axe < $axe_y_max ) ) {
    ligne($x1,$y1,$x2,$y2,$gris_foncé);
  }
  texte(3,$x1-2,$y1,$axe,$noir,"D","C");
}

// Dessin des barres.
$i = 0;
foreach($données as $clé => $valeur) {
  $i++;
  $x1 = $tx1 + $écart_barres + ($i-1)*($écart_barres+$largeur_barre);
  $x2 = $x1 + $largeur_barre;
  $y1 = $ty1;
  $y2 = $y1 + $valeur * $échelle_axe_y;
  rectangle($x1,$y1,$x2,$y2,$noir,$vert);
  texte(3,($x1+$x2)/2,$y1,$clé,$noir,"C","B");
}

// Générer l'image au format PNG = imagepng
// - soit à l'écran (pas de deuxième paramètre)
// - soit dans un fichier (deuxième paramètre = nom du fichier)
// Il existe des fonctions similaires pour d'autres formats.
imagepng($image /*, 'image.png' */);

// Supprimer l'image = imagedestroy
imagedestroy($image);

?>
```